

Robocup „Rescue Line“ Arduino Bausatz

Inhaltsverzeichnis

<u>1 Hardware Auswahl</u>	3
<u>1.1 Anforderungen</u>	3
<u>1.2 Bestandteile eines Roboters</u>	5
<u>1.3 Kommunikation zwischen Peripherien und Controller</u>	6
<u>1.3.1 Digital Out (digitalWrite)</u>	6
<u>1.3.2 PWM Out (analogWrite)</u>	6
<u>1.3.3 Digital In (digitalRead)</u>	7
<u>1.3.4 Analog In (analogRead)</u>	7
<u>1.3.5 SPI Bus</u>	8
<u>1.3.6 I2C Bus</u>	8
<u>1.4 Vorgehen bei der Hardware Planung</u>	10
<u>1.5 Übersicht Bauteile</u>	10
<u>1.6 I2C Adressen</u>	13
<u>1.7 SPI Chip>Selects</u>	14
<u>1.8 Stromaufnahme</u>	15
<u>2 Programmier-Umgebungen (IDEs)</u>	16
<u>2.1 Für Programmiersprache C++ mit Arduino API</u>	16
<u>2.1.1 Arduino (1.x)</u>	18
<u>2.1.2 Arduino IDE (2.x)</u>	21
<u>2.1.3 Visual Studio Code mit Arduino Extension</u>	24
<u>2.1.4 Problembehandlung</u>	27
<u>2.2 Für Programmiersprache Python mit Circuit Python API</u>	28
<u>2.2.1 Vorbereitungen</u>	28
<u>2.2.2 Arduino Lab for Circuit Python</u>	28
<u>2.2.3 Visual Studio Code mit Python Extension</u>	28
<u>3 Controller Vergleich</u>	29
<u>3.1 Arduino Due</u>	31
<u>3.1.1 Beispiel-Programme</u>	31
<u>3.1.2 Pinout</u>	31
<u>3.2 Arduino Nano RP2040 Connect</u>	33
<u>3.2.1 Beispiel-Programme</u>	33
<u>3.2.2 Pinout</u>	33
<u>3.2.3 Schaltplan</u>	34
<u>3.2.4 Bootloader laden und Betriebssystem flashen</u>	36
<u>3.2.5 Abstürze</u>	36
<u>3.3 OpenMV Cam H7</u>	37
<u>4 Bauteile</u>	38
<u>4.1 ZumoShield</u>	38
<u>4.2 DC-Motoren und Servos</u>	39
<u>4.2.1 2-Kanal DC-Motorsteuerung</u>	39
<u>4.2.2 2-Kanal I2C DC-Motorsteuerung</u>	42
<u>4.2.3 DC Motor</u>	42
<u>4.2.4 Servo Motor</u>	43
<u>4.3 User Interface</u>	46

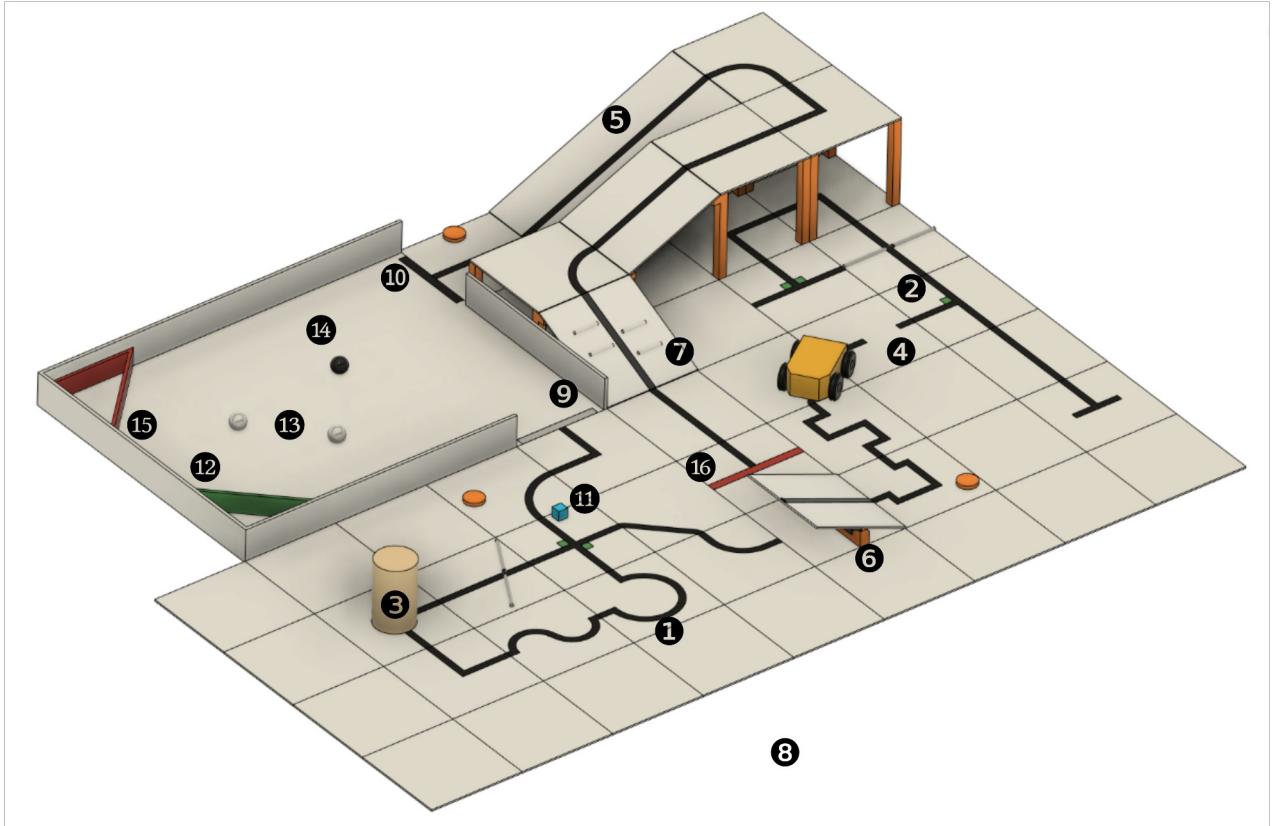
<u>4.3.1 Programmierbare RGB LED (extern)</u>	46
<u>4.3.2 I2C Displays</u>	46
<u>4.4 Optische Sensoren</u>	47
<u>4.4.1 OpenMV Cam</u>	47
<u>4.4.2 RGB Farbsensor</u>	47
<u>4.4.3 Spektrometer Farbsensor</u>	49
<u>4.4.4 6-Kanal Helligkeitssensor</u>	52
<u>4.5 Bewegungssensoren (IMUs)</u>	55
<u>4.5.1 9-Achsen IMU (Onboard ZumoShield)</u>	55
<u>4.5.2 6-Achsen IMU (Onboard Arduino Nano RP2040 Connect)</u>	55
<u>4.5.3 9-Achsen IMU mit Sensorfusion</u>	55
<u>4.6 Abstandssensoren</u>	57
<u>4.6.1 Mikro-Taster</u>	57
<u>4.6.2 1-Kanal Abstandssensor</u>	58
<u>4.6.3 64-Kanal Abstandssensor</u>	60
<u>4.7 Spannungsversorgung</u>	63
<u>4.7.1 LiPo Akku (7,4V)</u>	63
<u>4.7.2 3,3V Spannungsregler (Onboard Arduino)</u>	64
<u>4.7.3 3,3V Spannungsregler (Extern)</u>	64
<u>4.7.4 5V Spannungsregler (Onboard Arduino)</u>	65
<u>4.7.5 5V-Spannungsregler (Extern)</u>	65
<u>4.8 Sonstiges</u>	67
<u>4.8.1 I2C Multiplexer</u>	67
<u>4.8.2 Jumper-Kabel</u>	67
<u>4.8.3 Schalter</u>	67
<u>4.8.4 Platinen-Abstandshalter</u>	67

1 Hardware Auswahl

Hierbei geht es darum die Komponenten auszuwählen, die im Roboter verbaut werden sollen.

1.1 Anforderungen

Beispiel-Parcours (wird jedes Mal umgebaut):



(Stand Regeln 2023)

Ein Rescue-Roboter muss...

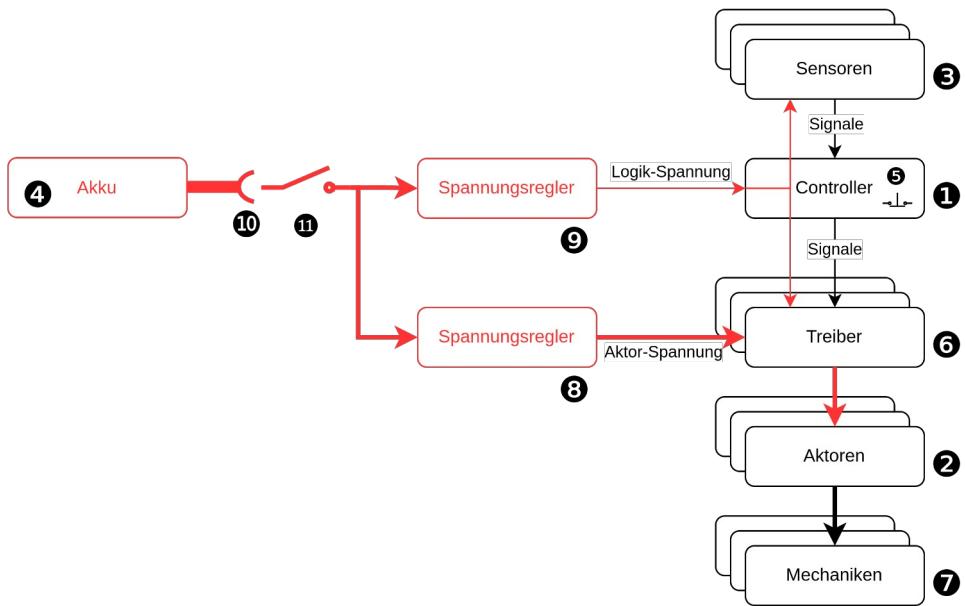
- mindestens...
 - fahren,
 - einer ① schwarzen Linie folgen und dabei nicht grün markierte Kreuzungen geradeaus überfahren,
 - an ② grünen Markierung abbiegen oder wenden,
 - ③ Hindernisse einseitig umfahren,
 - ④ Lücken in der Linie, ⑤ Rampen und ⑥ Wippen, ⑦ Bodenwellen und „Trümmer“ (z.B: Schaschlik-Spieße) überfahren, ohne dabei die Linie zu lange zu verlassen;
- in der vollen Ausbaustufe zusätzlich...
 - Sensorwerte vor dem Start des Wertungslaufs kalibrieren,

- Hindernisse beidseitig umfahren (wenn z.B. eine Wand oder ein **8** Abgrund im Weg ist),
- **9** Eingang (silberne, elektrisch leitfähige Markierung) und **10** Ausgang (schwarze Markierung) der Evakuierungszone erkennen,
- innerhalb der Evakuierungszone...
 - das **11** Rescue Kit (blauer Quader) in der **12** grünen Evakuierungszone abliefern (einzelnen oder zusammen mit **13** lebendigen Opfern) oder optional zusätzlich zuvor auf dem Parcours finden und aufsammeln,
 - **13** 2 lebendige Opfer (silberne elektrisch leitfähige Kugeln) finden und in der **12** grünen Evakuierungszone abliefern (einzelnen oder alle gleichzeitig),
 - **14** 1 totes Opfer (schwarze Kugel) erkennen und in der **15** roten Evakuierungszone abliefern,
- bei Wahl der Level 1 Evakuierungszone (Fläche auf dem Boden) reicht es, Rescue Kit und Opfer zu schieben, bei Level 2 (Kiste), müssen mindestens die Opfer aufgehoben werden (und wenn man das Rescue Kit auch aufsammeln will, auch dieses),
- am **16** Ende des Parcours (rote Markierung) anhalten.

Der Roboter muss **zwingend**

- einen Griff haben, mit dem er während eines Wertungslaufs aufgehoben und abgesetzt wird,
- einen Ein/Aus-Schalter haben, der das Programm bei einem „Lack of Progress“ (LoP) pausiert oder stoppt und bei einem (erneut) Versuch startet („Soft Reset“). Das darf ein Druck-Taster oder ein Kipp-Schalter sein und auch der Schalter, der den Akku abtrennt/verbindet („Hard Reset“). Es dürfen beide Schalter für Soft Reset und Hard Reset vorhanden und bei einem LoP verwendet werden.

1.2 Bestandteile eines Roboters



1 Controller: Das kann ein Mikrocontroller (z.B. Arduino Due) oder Einplatinen-Computer (z.B. Raspberry Pi) sein. Das Roboter-Programm läuft auf dem Controller, an den **2 Aktoren** und **3 Sensoren** angeschlossen sind (Logik-Spannung). **5** Ein Knopf/Taster wird empfohlen, um den Controller neu starten zu können („Soft Reset“). Dieser Knopf sollte gut zugänglich positioniert sein.

2 Aktoren: Alles, wo eine Information aus dem Controller herauskommt, also z.B. Motoren, Servos, Leucht-Dioden (LEDs), Displays. Für Aktoren benötigt man eine **6 Verstärkung** der Informationen aus dem Controller, weil sie viel Strom benötigen, die der Controller nicht liefern kann. Der Verstärker für einen Motor nennt sich Motortreiber. Ein Verstärker/Treiber ist sowohl mit der Logik-Spannung des Controllers als auch mit der **8 Hochleistungs-Spannungsversorgung** (**4 Akku**) verbunden. An Motoren und Servos werden meist **7 Mechaniken** wie Räder/Ketten/Greifer/Heber angebaut. LEDs und Displays nutzt man als User Interface.

3 Sensoren: Alles, wo Informationen in den Controller hineingehen, also z.B. Schalter/Taster, Photodioden. Sensoren können meist direkt an die **9 Logik-Spannungsversorgung** des Controllers angeschlossen werden, wenn es nicht zu viele werden - sonst muss man einen extra Spannungs-Regler verbauen, der die Spannung genauso herunter-spannt und reguliert wie die Logik-Spannung, aber mehr Strom liefern kann. Da die Positionierung der Sensoren enorm wichtig ist, hat auch dies einen Einfluss auf die Mechanik des Roboters.

4 Akku: Die Spannungsversorgung für den gesamten Roboter. Diese muss genug Strom liefern können, um alle Aktoren, Sensoren und den Controller zu betreiben. **9** Die Spannung muss für Controller und Sensoren stabilisiert und herunter-gespannt werden. Es ist empfohlen, auch die Spannung für Aktoren zu stabilisieren mit einem **8 Spannungs-Regler**. Der Akku muss austauschbar sein, benötigt also einen **10 Stecker**. Ideal ist auch ein **11 Ein/Aus Schalter**, der den kompletten Roboter spannungsfrei schalten kann, ohne den Stecker ziehen zu müssen. Stecker und Schalter müssen gut zugänglich positioniert werden und der Akku aufgrund seines großen Gewichts mittig unten im Roboter fixiert, damit er nicht herum rutscht.

1.3 Kommunikation zwischen Peripherien und Controller

Um Aktoren und Sensoren (Peripherien) mit dem Controller zu verbinden, gibt es verschiedene Möglichkeiten, die durch die gewählte Peripherie vorgegeben wird. Der Controller hat elektrische Anschlüsse (Pins), durch die Spannungs-Signale getrieben werden, meist vom Controller in die Peripherie, manchmal auch umgekehrt. Diese Signale repräsentieren Informationen. Nicht jede Art der Kommunikation geht an jedem Pin. Meist kann man für einen Pin zwischen mehreren Kommunikations-Arten wählen.

Welcher Pin was kann, erfährt man im „Pinout“ des Controllers

- [offizielles Pinout des Arduino Nano RP2040 Connect](#) oder Toms Pinout: [3.2 Arduino Nano RP2040 Connect](#)
- [offizielles Pinout des Arduino Due](#) oder Toms Pinout: [3.1 Arduino Due](#)
- Pinout OpenMV Cam: [3.3 OpenMV Cam H7](#)

1.3.1 Digital Out (digitalWrite)

Aus Controller-Perspektive: **Ausgang**

Beispiele: LED, Relais oder den Reset einer Peripherie ein- und ausschalten, die Drehrichtung eines Motors steuern, Basis für digitale Kommunikation

Der Controller stellt die Spannung am Pin entweder...

- auf 3,3V (Logikspannung) → elektrisch **HIGH**, logisch „Ja“, in C++ `1` oder `true`, in Python `1` oder `True`
- auf 0V (GND) → elektrisch **LOW**, logisch „Nein“, in C++ `0` oder `false`, in Python `0` oder `False`

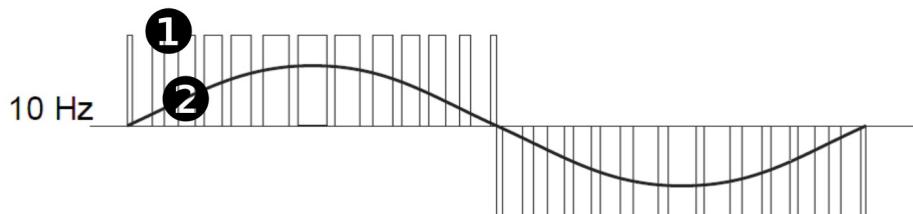
Das Signal ist binär (2 Zustände, digital). Damit kann man also Dinge ein- und ausschalten.

Das Ein- und Ausschalten erlaubt digitale Kommunikation (siehe auch [1.3.6 I2C Bus](#), [1.3.5 SPI Bus](#)). Z.B. lassen sich programmierbare RGB LEDs über ein Bit-Banging Protokoll über nur einen **Digital Out** steuern.

1.3.2 PWM Out (analogWrite)

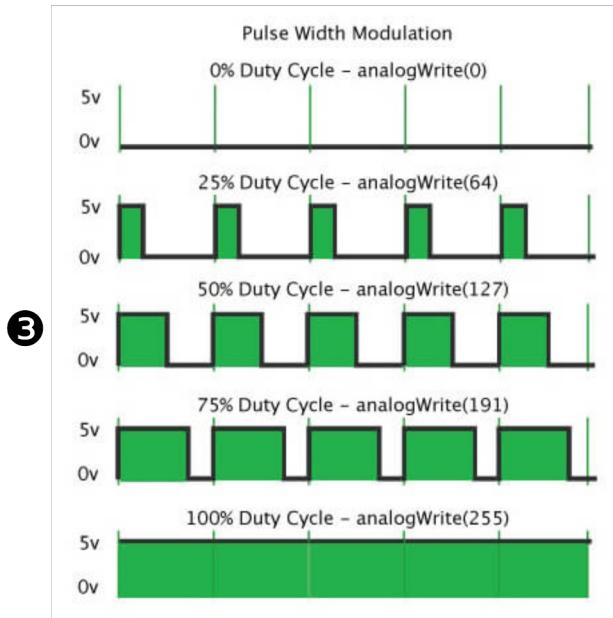
Aus Controller-Perspektive: **Ausgang**

Beispiele: Helligkeit von LEDs einstellen, Buzzer zum Piepsen bringen und die Geschwindigkeit von Motoren steuern



Spezialfall von [1.3.1 Digital Out \(digitalWrite\)](#), wobei der Pin **1** sehr schnell ein- und ausgeschaltet wird. Da die meisten analogen Peripherien träge sind, kommen sie nicht so schnell hinterher und zeigen nur den **2** Durchschnitt der Spannung über die Zeit. Aus Sicht einer LED

oder eines Motors ist dies also ein analoges Signal (es kann jede Spannung eingestellt werden), gesteuert durch das ③ Tastverhältnis.



Für das ① schnelle Ein- und Ausschalten könnte man auch einfach einen Digital Out Pin nehmen und ein Software-Programm schreiben, dass ihn schnell ein-/ausschaltet (Bit-Banging). Das würde den Prozessor aber sehr stark beschäftigen. Aus diesem Grund gibt es spezielle PWM Out Pins, denen man nur sagt, mit welcher Frequenz und welchem ③ Tastverhältnis (Duty Cycle) sie schalten sollen und tun dies automatisch.

1.3.3 Digital In (digitalRead)

Aus Controller-Perspektive: **Eingang**

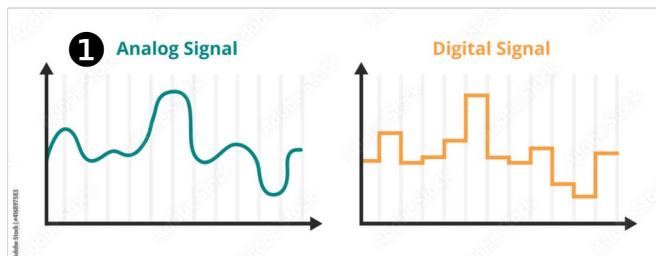
Beispiele: Zustand eines Schalters/Taster auslesen, Basis für digitale Kommunikation.

Es gibt die 2 Zustände wie bei [1.3.1 Digital Out \(digitalWrite\)](#). Nur die Richtung, in die die Information fließt, ist umgekehrt: Die Peripherie schaltet ein und aus und der Controller detektiert das.

1.3.4 Analog In (analogRead)

Aus Controller-Perspektive: **Eingang**

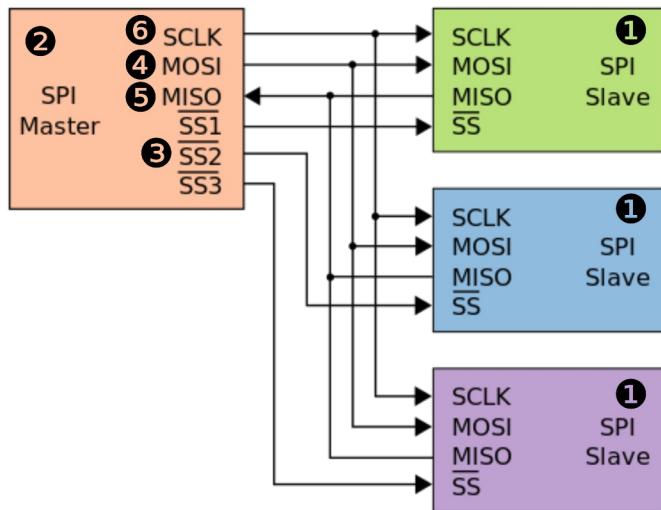
Beispiele: Den Foto-Strom einer Foto-IR-Diode messen, die Drehstellung eines Drehknopfes (Potentiometer) messen.



Hier misst der Controller ein echtes ① analoges Signal (es kann jede Spannung eingestellt werden), nicht nur 2 Zustände.

1.3.5 SPI Bus

Beispiele: Die Pixy Cam oder OpenMV Cam könnte man mit SPI ansprechen, aber wir nehmen lieber auch den [1.3.6 I2C Bus](#).

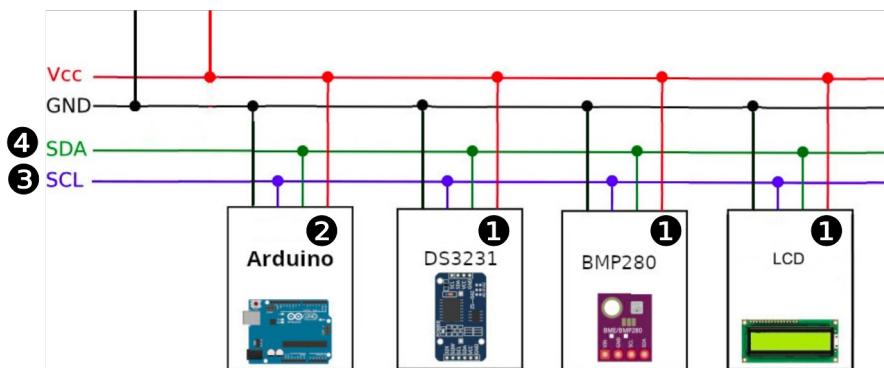


SPI ist ein Bus-System, d.h. man kann mehr als eine ① Peripherie/Slave an dieselben digitalen Pins des ② Controllers/Masters anschließen. Das spart Pins und der Controller kann kleiner sein. Dabei wird über einen Chip-Select Pin (③ CSx /SSx) eine Peripherie ausgewählt und dann Daten zur Peripherie gesendet (④ COPI /MOSI Pin). Diese antwortet dann mit Daten (⑤ CIPO/MISO Pin). Alle anderen Peripherien halten währenddessen die Klappe, weil deren Chip Select nicht ausgewählt wurde. Damit die Kommunikation exakt funktioniert, gibt der Master einen Takt (⑥ SCK/SCLK Pin) vor, in dem Signale zwischen **HIGH** und **LOW** umgeschaltet werden dürfen.

Man kann also nur eine Peripherie nach der anderen ansprechen und nie gleichzeitig, weil sie nicht gleichzeitig antworten dürfen (nur, wenn sie dran sind). Das ist aber nicht schlimm, da ein Single-Thread-Programm die Sensoren eh nicht parallel auslesen kann, sondern nur nacheinander.

1.3.6 I2C Bus

Beispiele: Alle Rescue Sensoren sind I2C-Sensoren. Es gibt aber auch I2C-Motorsteuerungen und Displays.



Ähnlich wie der [1.3.5 SPI Bus](#), spart aber noch mehr Pins. Es gibt wieder einen gemeinsamen Takt (③ SCL Pin), den der ② Controller/Master vorgibt. Der Master sendet erst Daten (④ SDA Pin) und dann antwortet die ① Peripherie/Slave auf selben Leitung (auch ④ SDA Pin). Es gibt keine

Chip-Select-Pins, sondern jeder Slave hat eine **Adresse**. Die Adressierung erfolgt über das Daten-Paket, dass der Master an den Slave sendet. Alle Slaves lauschen immer auf SDA. Wenn ein Slave seine Adresse in den Daten auf SDA erkennt, lauscht er auch auf die Daten und antwortet dann. Alle anderen halten in der Zeit die Klappe.

Aus diesem Grund ist es sehr wichtig Adress-Konflikte zu vermeiden, d.h. man darf nicht 2 Peripherien mit derselben Adresse an denselben Bus anschließen, weil sie gleichzeitig antworten würden!

Hausmittel gegen I2C Address-Konflikte:

- **mehrere I2C Busse**, d.h. die Peripherien, die gleiche Adressen haben, an unterschiedliche Pins anschließen (muss der Controller können und man braucht mehr Pins)
 - **Software I2C**, d.h. anstatt die I2C-fähigen Pins des Controllers zu verwenden, nimmt man beliebige Digital In/Out Pins und benutzt eine Software I2C Bibliothek (geht immer, aber Bit-Banging ist langsamer als die echten Hardware I2C Pins und man muss die Libraries anfassen. **Also lieber vermeiden.**)
- **I2C Multiplexer**, d.h. ein weiteres Bauteil mit zusätzlichen Chip-Select Pins, dass zwischen Master und die Slaves geschaltet wird, also quasi aus I2C SPI machen (kann jeder Controller, braucht mehr Pins)
- Peripherien wählen, die einen zusätzlichen Pin haben, mit denen man sie temporär im **Reset** hält und damit abschaltet, was auch wie ein Chip-Select wirkt. Bei manchen dieser Peripherien kann man auch die Adresse temporär programmieren (ändern). Manchmal klappt es auch die Adressen ohne einen zusätzlichen Reset Pin umzaprogrammieren.

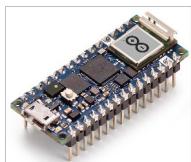
1.4 Vorgehen bei der Hardware Planung

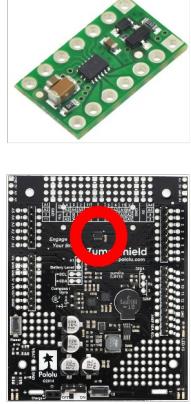
1. Regeln studieren (siehe [1.1 Anforderungen](#)).
2. Überlegen, mit welchen Bauteilen man die Aufgaben aus den Regeln umsetzen könnte. Bauteile wählen (siehe [1.5 Übersicht Bauteile](#)) und in Wunschliste schreiben.
3. I2C Bauteile den 2 zur Verfügung stehenden I2C Bussen zuordnen. Dabei vermeiden, dass I2C Adress-Konflikte entstehen. Wenn man einen I2C Sensor mehr als 2x verwenden will, werden entweder zusätzliche RESET Pins notwendig oder ein I2C Multiplexer (siehe [1.6 I2C Adressen](#)).
4. Benötigte Pins nach Typ gruppiert (siehe [1.3 Kommunikation zwischen Peripherien und Controller](#)) durchzählen (siehe [1.5 Übersicht Bauteile](#)) und darauf achten, dass SPI Bauteile zusätzliche Pins benötigen [1.7 SPI Chip-Selects](#)).
5. Controller wählen (siehe [3 Controller Vergleich](#)). Prüfen, ob genügend Pins zur Verfügung stehen. Wenn zu wenig, kann ein Bauteil mit vielen Digital I/O vielleicht durch ein I2C Bauteil ersetzt werden oder es muss der Controller gewechselt werden.
6. Stromaufnahme für die jeweiligen Spannungs-Ebenen aufsummieren (siehe [1.8 Stromaufnahme](#)). Wenn das Strom-Budget des Controllers überschritten wird, müssen zusätzliche Spannungsregler vorgesehen werden. Man benötigt immer einen Spannungsregler für DC-Motoren und Servos (bei ZumoShield onboard).
7. Schaltplan erstellen (siehe Beispiele in 4 Bauteile).
8. Layout erstellen (siehe Beispiele in 4 Bauteile).

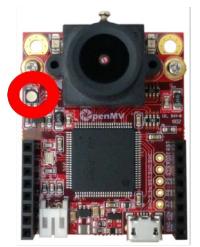
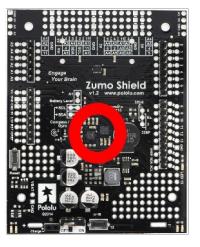
1.5 Übersicht Bauteile

Folgende Bauteile stehen zur Verfügung:

Pins, die mit dem Controller verbunden werden müssen und etwas vom Pin Budget des Controllers verbrauchen, sind mit **✓** markiert.

Kategorie	Name	Pins (✓ Kommunikation)	Bild
Controller	3.1 Arduino Due 3,3V Logik, 5V Pin: 800mA, 3,3V Pin: 800mA, viele Pins, spart Bauteile zusammen mit ZumoShield (Spannungsregler, Motortreiber)	Pin Budget: 2 I2C Bus 4 SPI Bus 54 Digital In/Out, davon... <ul style="list-style-type: none">• 1 durch LED belegt• 12 PWM Out fähig• 12 SPI fähig• 12 Analog In fähig<ul style="list-style-type: none">◦ davon 4 I2C fähig	
	3.2 Arduino Nano RP2040 Connect 3,3V Logik, 5V Pin: X , 3,3V Pin: 800mA, Bluetooth Debugging möglich,	Pin Budget: 2 I2C Bus 1 SPI Bus 20 Digital In, davon... <ul style="list-style-type: none">• 18 Digital Out• 1 durch LED belegt	

	<p>sehr kompakt, Python möglich, Steckbrett-Entwicklung möglich</p> <ul style="list-style-type: none"> • 18 PWM Out fähig • 3 SPI fähig • 6 Analog In fähig <ul style="list-style-type: none"> ◦ davon 4 I2C fähig 		
	<p>3.3 OpenMV Cam H7 max. 5V Versorgungsspannung → benötigt externen Spannungsregler, onboard Kamera, 3,3V Logik, 5V Pin: X, 3,3V Pin: 250mA, nur Python möglich</p>	<p>Pin Budget:</p> <p>2 I2C Bus 1 SPI Bus 10 Digital In/Out, davon...</p> <ul style="list-style-type: none"> • 10 PWM Out fähig <ul style="list-style-type: none"> ◦ 3 Servo fähig • 3 SPI fähig • 4 I2C fähig • 1 Analog In fähig 	
Aktor	<p>4.2.1 2-Kanal DC-Motorsteuerung DRV8835 (1x Onboard bei 4.1 ZumoShield) vorwärts, rückwärts, Geschwindigkeit, sehr schnelle Kommunikation, Logik-Spannung: 2..7V Motor-Spannung: 0..11V Strom pro Kanal: 1,2A → 1 Motor pro Kanal, sonst reicht der Strom nicht</p>	<p>Beide Motoren drehen unterschiedlich: ▶▶ 2x Digital Out ▶▶ 2x PWM Out</p> <p>Beide Motoren drehen gleich: ▶ 1x Digital Out ▶ 1x PWM Out</p>	
	<p>4.2.2 2-Kanal I2C DC-Motorsteuerung vorwärts, rückwärts, Geschwindigkeit, langsamere Kommunikation, Logik-Spannung: 3,3V Motor-Spannung: 3..11V Strom pro Kanal: 1,2A</p>	<p>▶▶ SDA, SCL (I2C Adresse: 0x5A, konfigurierbar)</p>	
	<p>4.2.3 DC Motor Spannung: 6V, Stromaufnahme: max. 1,6A, vorwärts, rückwärts, Geschwindigkeit</p>	<p>mit einem Motorsteuerungs-Kanal verbunden (für Ketten braucht man 2, für Räder braucht man 2 oder 4)</p>	
	<p>4.2.4 Servo Motor Spannung: 6V, Stromaufnahme: max. 800mA, hält die Position (für Heben, Greifen, Klappen)</p>	<p>▶ 1x PWM Out</p>	

	<p>RGB LED</p> <p>LED, deren Farbe über 3 Farbkanäle (PWM) eingestellt werden kann</p>	<p>Onboard bei 3.2 Arduino Nano RP2040 Connect (3 zusätzliche PWM Out)</p> <p>Onboard bei 3.3 OpenMV Cam H7 (4 zusätzliche PWM Out)</p>	 
Sensor	<p>6-Achsen Bewegungssensor LSM6DSOX</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> 3 Achsen Beschleunigung, <input checked="" type="checkbox"/> 3 Achsen Gyro, <input checked="" type="checkbox"/> 3 Achsen Magnetfeld (Kompass), <input checked="" type="checkbox"/> Sensor-Fusion 	<p>Onboard bei 3.2 Arduino Nano RP2040 Connect (I2C Adresse: 0x6A)</p>	
	<p>9-Achsen Bewegungssensor (LSM303D und L3GD20H)</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> 3 Achsen Beschleunigung, <input checked="" type="checkbox"/> 3 Achsen Gyro, <input checked="" type="checkbox"/> 3 Achsen Magnetfeld (Kompass), <input checked="" type="checkbox"/> Sensor-Fusion 	<p>Onboard bei 4.1 ZumoShield (I2C Adressen: 0x1D, 0x6B - beide belegt)</p>	
	<p>4.5.3 9-Achsen IMU mit Sensorfusion (BNO055)</p> <ul style="list-style-type: none"> <input checked="" type="checkbox"/> 3 Achsen Beschleunigung, <input checked="" type="checkbox"/> 3 Achsen Gyro, <input checked="" type="checkbox"/> 3 Achsen Magnetfeld (Kompass), <input checked="" type="checkbox"/> Sensor-Fusion 	<p>» SDA, SCL (I2C Adresse: 0x28)</p>	
	<p>4.6.1 Mikro-Taster “Abstandssensor“ Messbereich: 0mm</p>	<p>» 1x Digital In</p>	
	<p>4.6.2 1-Kanal Abstandssensor VL53L0X schnell und genau Messbereich: 50mm..1200mm</p>	<p>» SDA, SCL (I2C Adresse: 0x29, programmierbar)</p>	
	<p>4.6.3 64-Kanal Abstandssensor VL53L5CX fortgeschrittene Objekterkennung (3D Punktwolke), aber langsamer Messbereich: 0..4000mm</p>	<p>» SDA, SCL (I2C Adresse: 0x29, programmierbar)</p>	
	<p>3.3 OpenMV Cam H7 als „Kamera-Sensor“ fortgeschrittene Erkennung von Farben, Linien und Kreisen (RGB Pixel), eigener Controller: Es muss ein Bilderkennungs-Programm geschrieben werden.</p>	<p>» SDA, SCL (I2C Adresse: frei programmierbar)</p>	

	4.4.4 6-Kanal Helligkeitssensor „ZumoShield front expansion“ misst die Intensität von reflektiertem Licht, also die Helligkeit	■■■■■■ 6x Digital I/O (auf 4.1 ZumoShield Pins 5, A2, A0, 11, A3, 4)	
	4.4.2 RGB Farbsensor TCS34725 misst rot, grün und blau Anteile im Licht, Farbe muss das Programm selbst ausrechnen	■■ SDA, SCL (I2C Adresse: 0x29)	
	4.4.3 Spektrometer Farbsensor AS7262 misst nur auf bestimmten Frequenzen, gibt einem die Farbe direkt aus	■■ SDA, SCL (I2C Adresse: 0x49)	
Sonstiges	4.8.1 I2C Multiplexer erlaubt denselben I2C Sensor mehrmals zu verwenden, spart zusätzliche SDA, SCL, wenn man sonst mehrere I2C Busse bräuchte	■■ SDA, SCL, ■... log2(Anzahl I2C Slaves)x Digital Out	
	Bluetooth Low Energy (BLE) / WiFi Im Wettbewerb nicht legal, aber sehr praktisch fürs Testen/Debuggen.	Onboard bei 3.2 Arduino Nano RP2040 Connect (interner UART (Serial3))	
Spannungsversorgung	Akkuspannungs-Anzeige damit man weiß, wann der Akku gewechselt werden muss, verfügbare Farben: gelb, grün, rot, blau	VIN, GND (vom Akku)	
	4.7.1 LiPo Akku (7,4V) Spannung: 4,7V Strom: 37A	VIN, GND (Versorgung für alles)	
	Akku-/Hard Reset Schalter (Onboard bei 4.1 ZumoShield)	VIN (vom Akku)	
	Spannungsregler 3,3V Versorgung für Sensoren	VIN, GND (vom Akku)	TBD
	Spannungsregler 5V Versorgung für Aktoren und 4.4.4 6-Kanal Helligkeitssensor	VIN, GND (vom Akku)	TBD

1.6 I2C Adressen

Regeln für die Zuordnung der Bauteile zu I2C Bussen (siehe auch [1.3.6 I2C Bus](#)):

- 2 Bauteile mit derselben I2C Adresse dürfen nicht an denselben I2C Bus angeschlossen werden,
- es sei denn, die Adresse ist programmierbar (änderbar):

- Bei manchen Bauteilen kann man die Adresse programmieren, ohne andere Chips währenddessen abzuschalten (im RESET zu halten), bei anderen nicht.

Prozedur I2C Adresse umprogrammieren:

Situation: Peripherie A und Peripherie B mit derselben Adresse am selben Bus. Peripherie A hat einen zusätzlichen RESET Pin, der via Digital Out mit dem Arduino verbunden ist.

- Peripherie A in den RESET zwingen und damit abschalten. Die antwortet jetzt nicht mehr.
- Peripherie B Adresse ändern (via I2C ohne Konflikt möglich, da A abgeschaltet ist).
- Peripherie A den RESET wieder loslassen. Beide Peripherien sind gebootet und haben unterschiedliche Adressen.

Bei manchen Kombinationen von Bauteilen können Schritt 1 und 3 übersprungen werden.

Die geänderte Adresse wird nicht abgespeichert, muss also immer wieder gesetzt werden, wenn die Peripherie einen Spannungsausfall hatte oder im RESET war.

Bauteil	Bus	Adresse (7-bit)	Kommentar
OpenMV Cam	wählbar	wählbar	wir schreiben das Programm; wir wählen die Adresse
I2C Motorsteuerung	wählbar	z.B. 0x5A	konfigurierbar über Lötbrücken
Arduino Nano RP2040 Connect Onboard: • LSM6DSOX • ATECC608A	I2C0 (Wire) I2C0 (Wire)	0x6A 0x60	Auf dem Board festgelötet. Können wir also nicht entfernen.
ZumoShield Onboard: • LSM303D • L3GD20H	I2C0 (Wire) I2C0 (Wire)	0x1D 0x6B	Auf dem Board festgelötet. Können wir also nicht entfernen.
9-Achsen-Bewegungssensor	wählbar	0x28	
AS7262	wählbar	0x49	
VL53L0X	wählbar	0x29	programmierbar, RESET Pin verfügbar
VL53L5CX	wählbar	0x29	programmierbar, RESET Pin verfügbar
TCS34725	wählbar	0x29	zusammen mit VL53L0X und VL53L5CX ohne zusätzliche RESET Pins verwendbar

1.7 SPI Chip-Selects

Für jedes SPI-Bauteil am selben SPI Bus muss ein eigener Chip-Select Pin vorgesehen werden (Digital Out). Siehe auch [1.3.5 SPI Bus](#).

1.8 Stromaufnahme

Regeln für die Entscheidung, ob ein zusätzlicher Spannungsregler notwendig wird:

- Digitale Logik darf nie aus dem Akku direkt versorgt werden, sondern die Spannung muss immer herunter-gespannt und stabilisiert werden. Das macht ein Spannungsregler.
- Wenn ein Bauteil nur weniger Maximalspannung als die des Akku verträgt (7,4V), muss die Spannung herunter-gespannt werden. Das macht ein Spannungsregler.
- Wird das Strom-Budget des Arduino Onboard Spannungsreglers überschritten (siehe [3 Controller Vergleich](#)), muss ein externer Spannungsregler vorgesehen werden, der mehr Strom liefern kann.
- DC-Motoren und Servos müssen eigene Spannungsregler verwenden oder direkt an den Akku ohne Spannungsregler angeschlossen werden, da sie sonst die digitale Logik stören.

Bauteil	Stromaufnahme		
	3,3V	6V	Akku (7,4V)
OpenMV Cam	160 mA	-	-
I2C Motorsteuerung	???	-	siehe „DC-Motor“
Arduino Nano RP2040	???	-	-
ZumoShield Onboard	-	-	>20 mA
9-Achsen-Bewegungssensor	50 mA	-	-
I2C Multiplexer	???	-	-
AS7262	38 mA	-	-
VL53L0X	19 mA	-	-
VL53L5CX	150 mA (bei Tests sehr stromhungrig)	-	-
TCS34725	38 mA	-	-
Servo-Motor	-	max. 800 mA	-
DC-Motor	-	max. 1500 mA	
RGB LED	max. 60 mA	-	-
Reflektions-Sensorleiste	-	240 mA	-

2 Programmier-Umgebungen (IDEs)

Um frustrationsarm und ergonomisch zu arbeiten, ist eine Integrated Development Environment (IDE) enorm wichtig. Sicherlich kann man textbasierten Quellcode in jedem Text-Editor schreiben, aber IDEs haben idealerweise alle Knöpfe in einem Fenster und weisen einen auf Fehler hin, bevor man den Quellcode ausprobiert:

1. **Projekt bauen:** Zumindest bei der Programmiersprache C++ muss der Quellcode in Maschinencode übersetzt werden. Bei Python kann dieser Schritt entfallen.
2. **Board flashen:** Den Maschinencode auf das Board (z.B. den Arduino) kopieren.
3. **Boards installieren:** Die Punkte (1) und (2) unterscheiden sich mit jedem Arduino Board, also muss die sogenannte „Toolchain“ für den Arduino (via Boardverwalter) installiert werden.
4. **mehrere Projekte in einem Fenster:** Das ist sehr praktisch, um schnell zwischen Beispielprogrammen und dem Roboter-Hauptprogramm hin- und herzuschalten und sie miteinander zu vergleichen.
5. **Serial Monitor:** Textausgabe, mit der man sich anzeigen lassen kann, was das Programm gerade macht.
6. **Serial Plotter:** Nutzt die Textausgabe vom Serial Monitor, um Diagramme anzuzeigen.
7. **Syntax Highlighting:** Sehr nützlich, um auch von Weitem z.B. eine Zahl von einem „String“ zu unterscheiden oder eine `[` eckige Klammer vor einer `{` geschweiften Klammer. Dafür werden diese Syntax-Bausteine unterschiedlich eingefärbt.
8. **Code Completion:** Man muss nicht alles auswendig wissen und komplett ausschreiben. Die IDE macht nützliche Vorschläge, die man mit einem Tastendruck übernehmen kann.
9. **Linting:** Die IDE erkennt bereits Syntax-Fehler, die sonst beim Bauen auftreten würden und markiert diese. Das spart Zeit und man muss nicht kryptische Compiler-Fehler verstehen.
10. **Refactoring:** Man hat eine Variable an X Stellen verwendet und will sie umbenennen. Refactoring ist ein Feature, wobei man die Variable an nur einer Stelle umbenennt und die IDE benennt sie überall automatisch mit um.
11. **Debugging:** Man kann den Code jederzeit anhalten und sich anschauen, wo das Programm gerade ist und welchen Wert welche Variablen haben. Und das ganz ohne Serial Monitor.

Für unterschiedliche Programmiersprachen gibt es unterschiedliche IDEs.

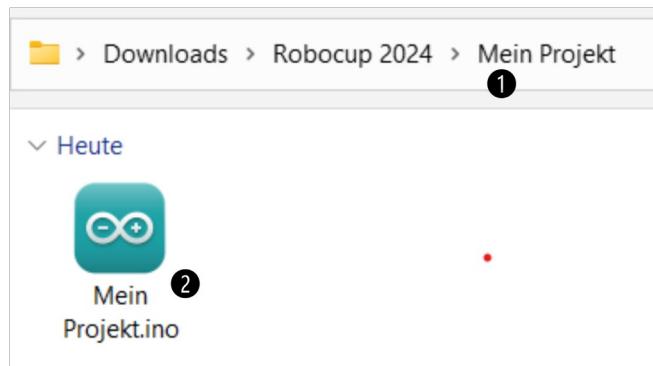
2.1 Für Programmiersprache C++ mit Arduino API

Wenn wir von „Arduino“ sprechen, meinen wir eigentlich die Programmiersprache C++ und einer Library, die „Arduino API“, die solche Dinge wie `setup()`, `loop()` und `Serial.println()` zur Verfügung stellt. Diese API gibt es für alle Arduinos. Dafür gibt es mehrere Editoren. Diese als „IDE“ zu bezeichnen, ist ein bisschen großzügig, aber wir belassen es mal dabei:

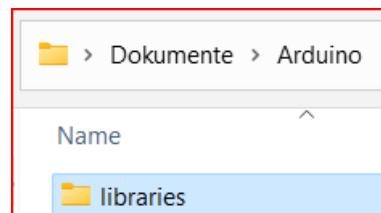
Feature	Arduino 1.x	Arduino IDE 2.x / Arduino CLI	Visual Studio Code
Boards installieren	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/> X
Projekt bauen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> (via Arduino CLI)
Board flashen	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> (via Arduino CLI)
mehrere Projekte in einem Fenster	<input type="checkbox"/> X	<input type="checkbox"/> X	<input checked="" type="checkbox"/> (man muss vor dem Bauen/Flashen das Projekt auswählen)

Serial Monitor	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> (kann auch gestartet und gestoppt werden, ohne das Fenster schließen zu müssen)
Serial Plotter	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> (schöner)	<input type="checkbox"/>
Syntax Highlighting	<input checked="" type="checkbox"/> (rudimentär)	<input checked="" type="checkbox"/> (schöner)	<input checked="" type="checkbox"/> (schöner)
Code Completion	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> (wenn der <code>.vscode</code> Ordner richtig konfiguriert ist)
Linting	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/> (wenn der <code>.vscode</code> Ordner richtig konfiguriert ist)
Refactoring	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Debugging	<input type="checkbox"/>	<input checked="" type="checkbox"/> (wenn man einen Debugger kauft)	<input checked="" type="checkbox"/> (wenn man einen Debugger kauft)

Egal welche IDE, ein Arduino Projekt besteht immer aus einem **1** Ordner und einer **2** `.ino` Datei, die genauso heißt wie der Ordner. Zusätzlich können natürlich noch andere Dateien im selben Ordner liegen, aber die `.ino` Datei ist der Einstiegspunkt des Programms und muss alle anderen Quellcode-Dateien importieren.



Egal welche IDE, Libraries müssen eigentlich immer in den `Eigenen Dateien\Documents\Arduino` installiert werden. Das werden wir später ändern:



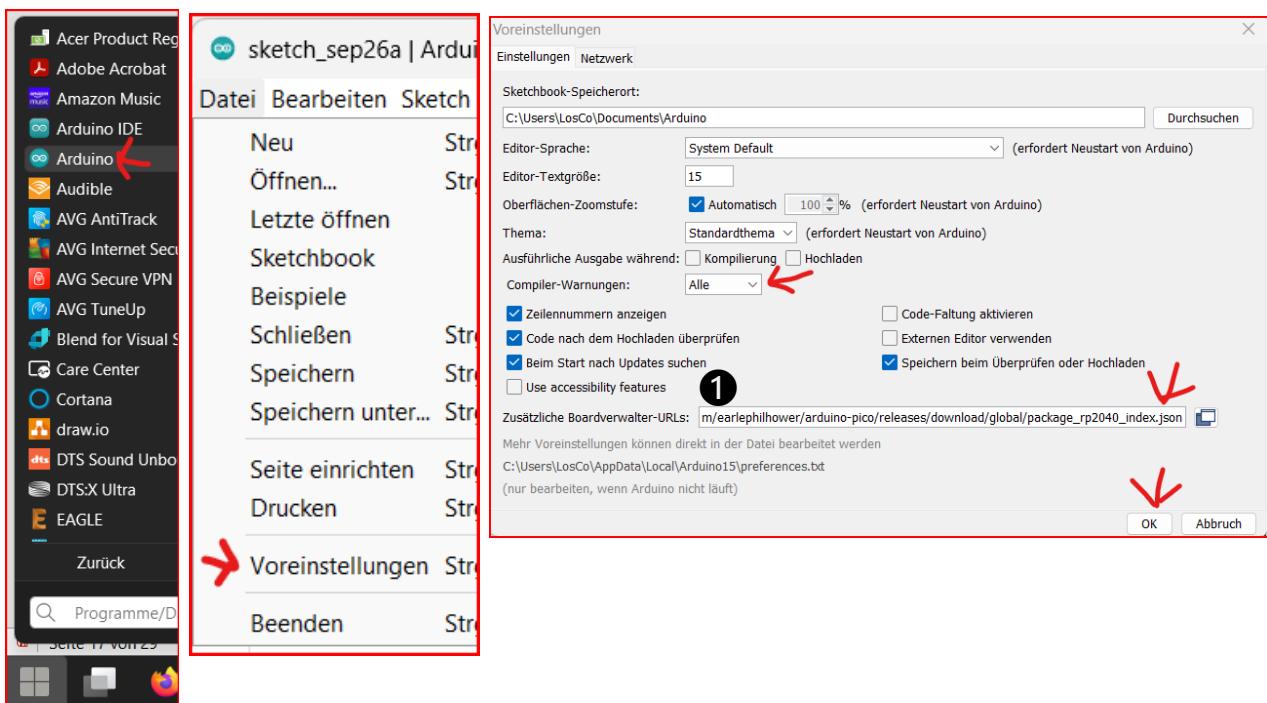
2.1.1 Arduino (1.x)

Download und Installation: <https://www.arduino.cc/en/software>

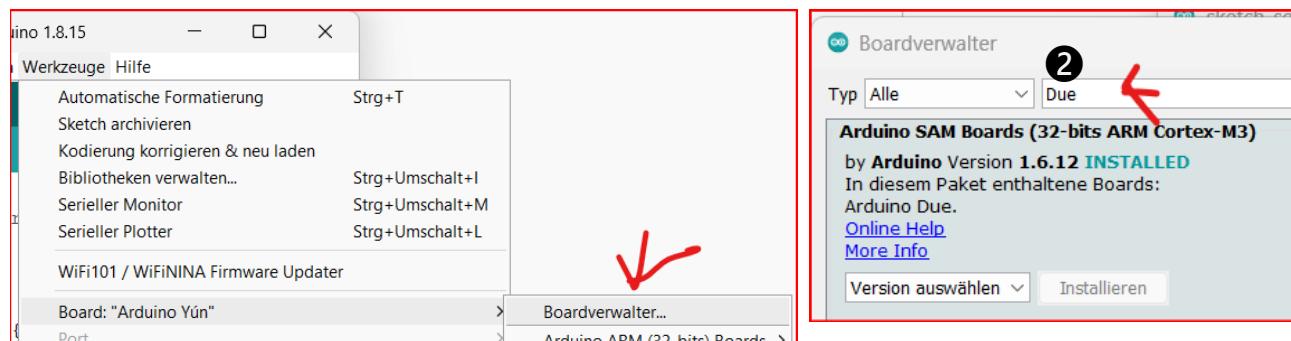
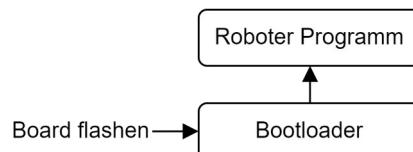


Boards installieren: (Arduino Due und Arduino Nano RP2040 Connect):

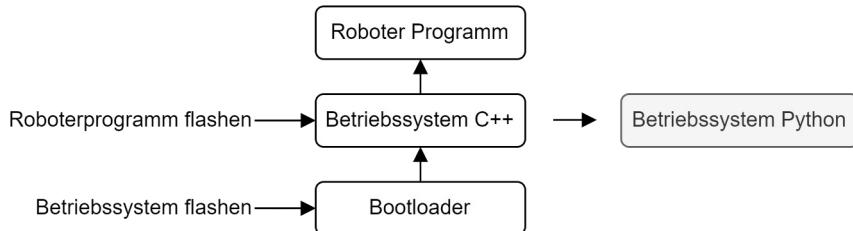
- 1 Zusätzliche Boardverwalter URL eintragen: https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json



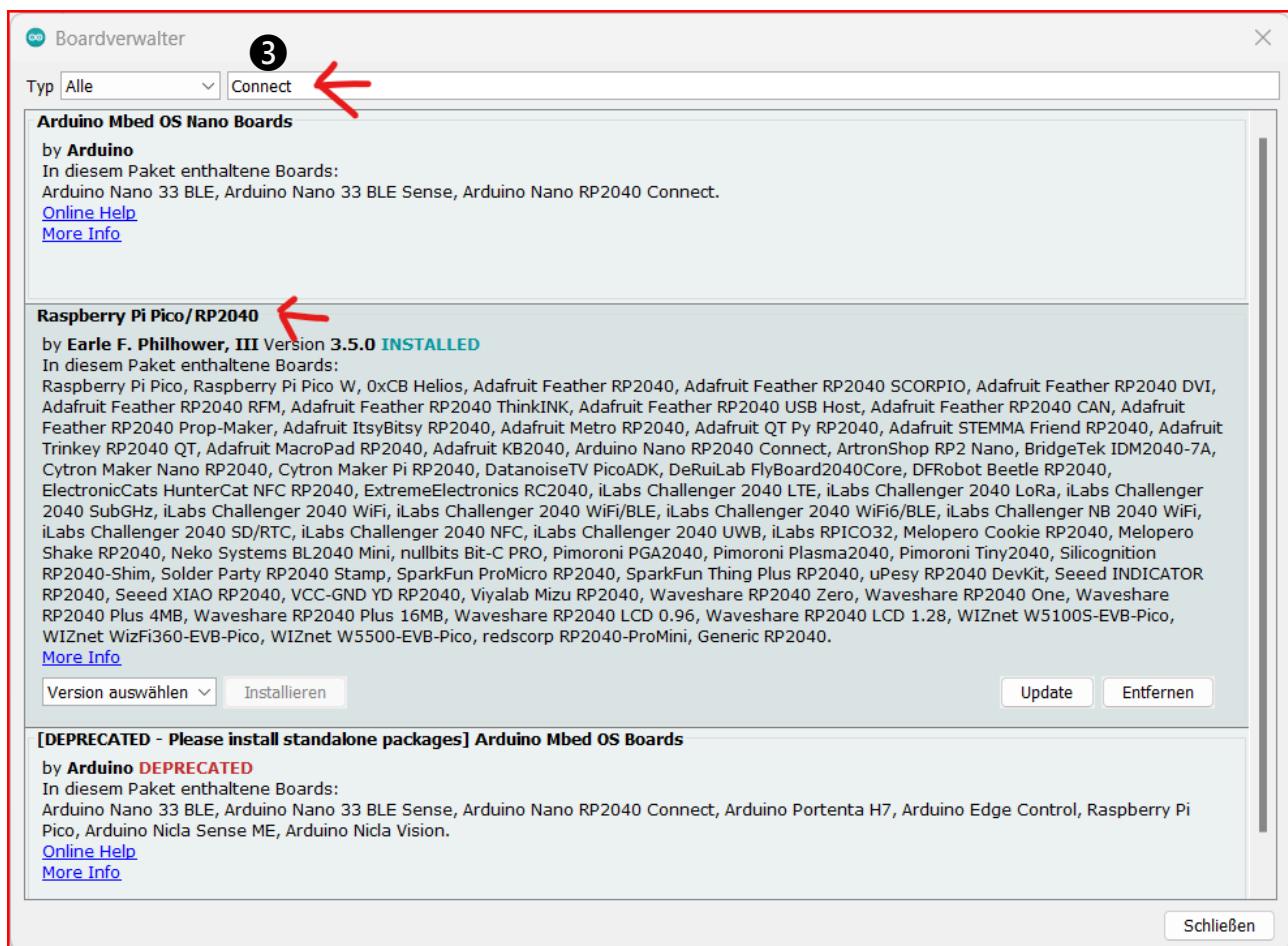
- 2 Arduino Due installieren. Der Due hat zwar einen Bootloader, aber kein Betriebssystem (OS). Das Roboter-Programm läuft also „Bare Metal“:



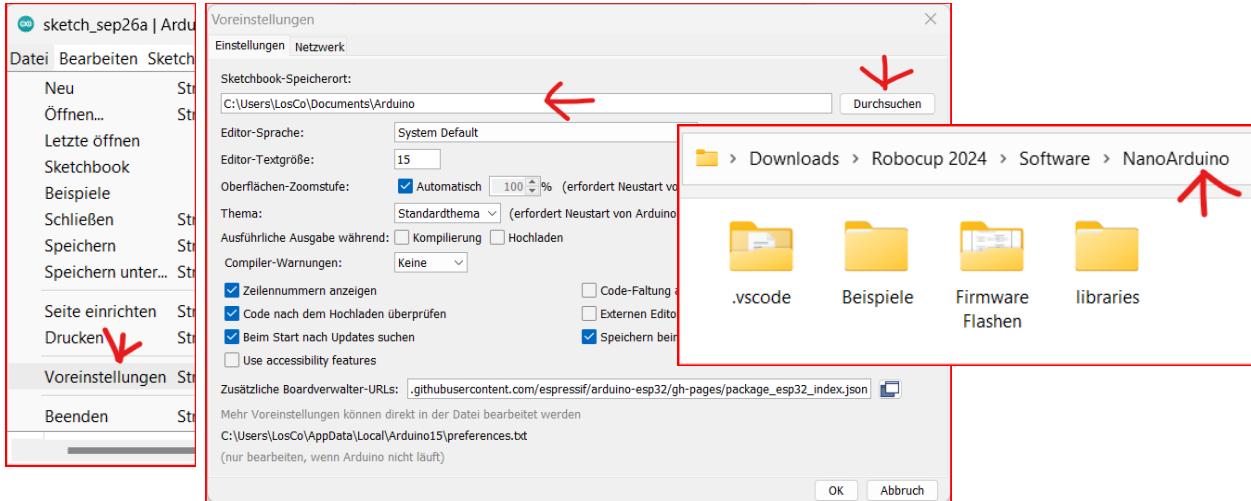
- 3 Arduino Nano RP2040 Connect installieren. Dieser Arduino ist in Wirklichkeit ein kleiner Raspberry Pi und hat somit ein Betriebssystem (OS). Das Betriebssystem kann man austauschen und somit eins für C++ oder Python draufspielen:



Wir benutzen nicht das offizielle OS von Arduino, sondern ein verbessertes für den Raspberry Pi Pico. Nur mit diesem OS bekommt man einen zweiten I2C Bus:

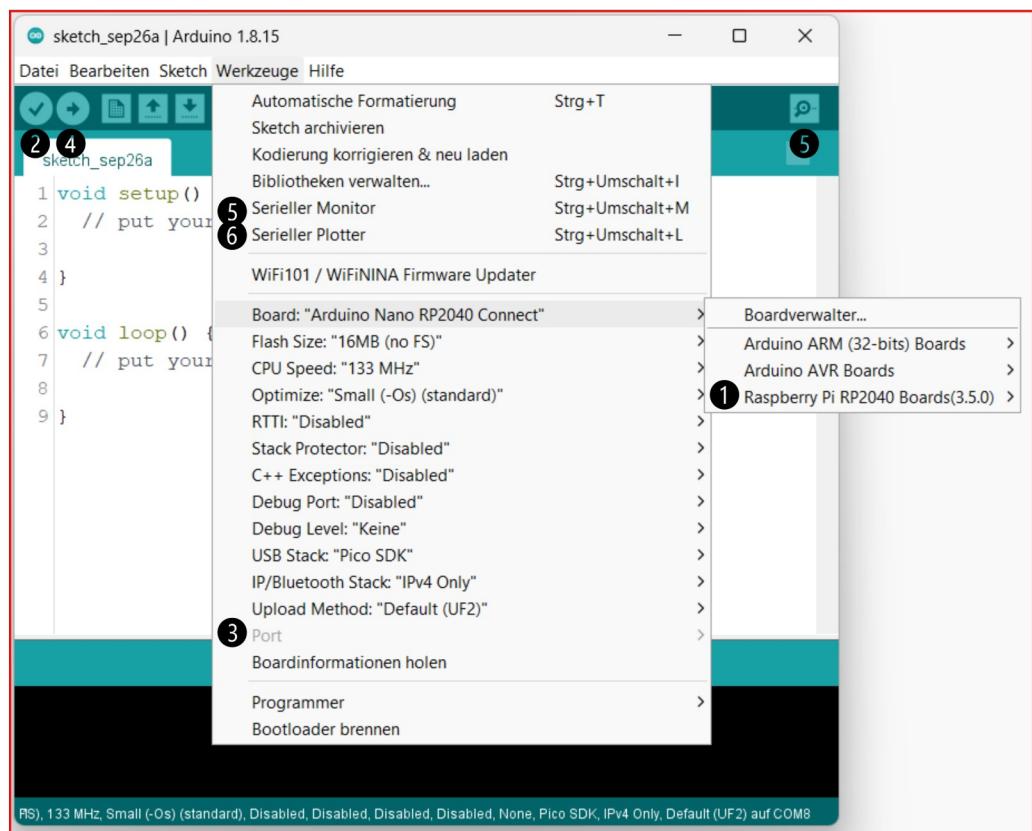


C++ Workspace öffnen: Wir wollen, dass unsere Programme portabel sind, d.h. Libraries sollen nicht in den eigenen Dateien liegen. Die Programme sollen auf einem USB Stick funktionieren und auf den Laptops beim Wettbewerb. Man kann den Ordner konfigurieren, wo sich die Libraries befinden. Im Robotik Tausch liegt ein Beispiel-Workspace namens `ArduinoNano`. Diesen auswählen. Da liegt unser `libraries` Ordner drin.



Jetzt kann man alle .ino Dateien des Ordners `ArduinoNano` öffnen (Beispielprogramme oder eigene Programme). Es öffnet sich dafür immer ein neues Arduino Fenster.

- 1 Board auswählen (1 mal, bevor man anfängt zu arbeiten)
- 2 Projekt bauen
- 3 Port auswählen (jedes Mal, nachdem man den Arduino vom USB getrennt hat)
- 4 Board flashen
- 5 Serial Monitor
- 6 Serial Plotter



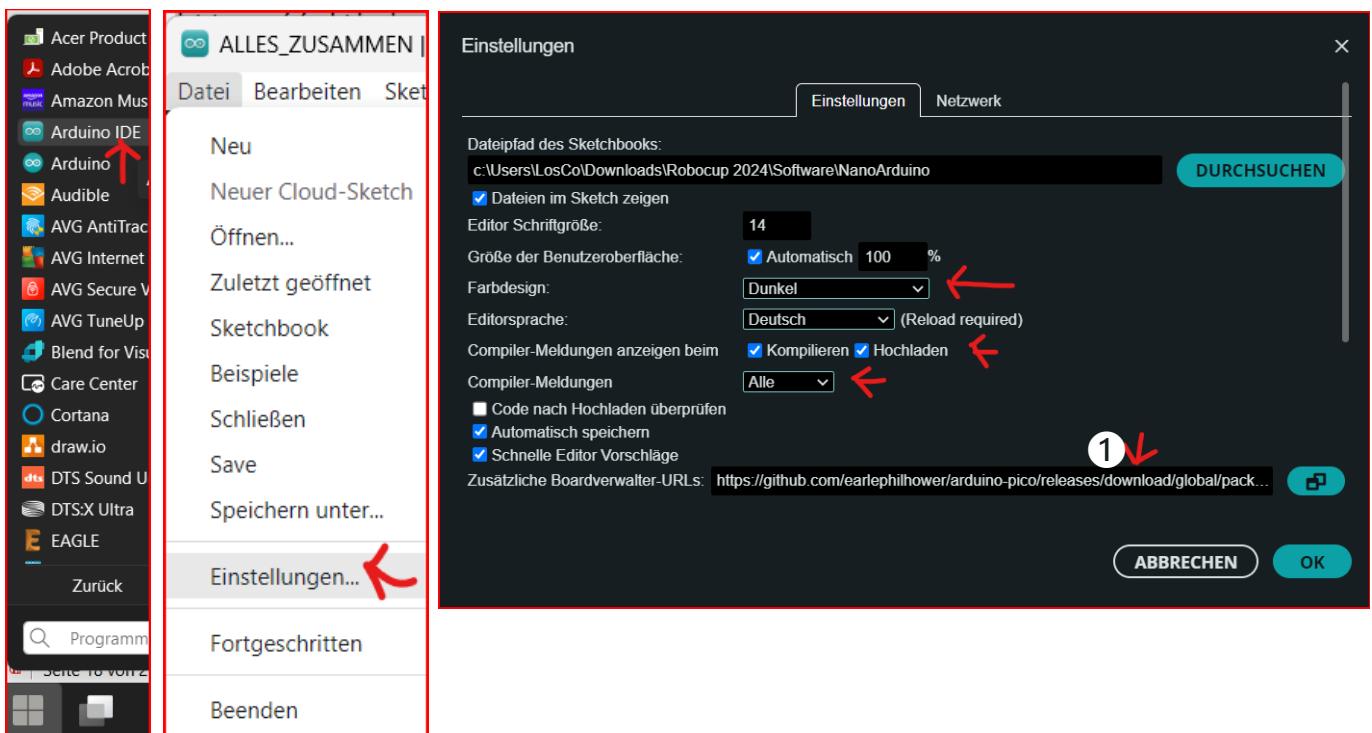
2.1.2 Arduino IDE (2.x)

Download und Installation: <https://www.arduino.cc/en/software>

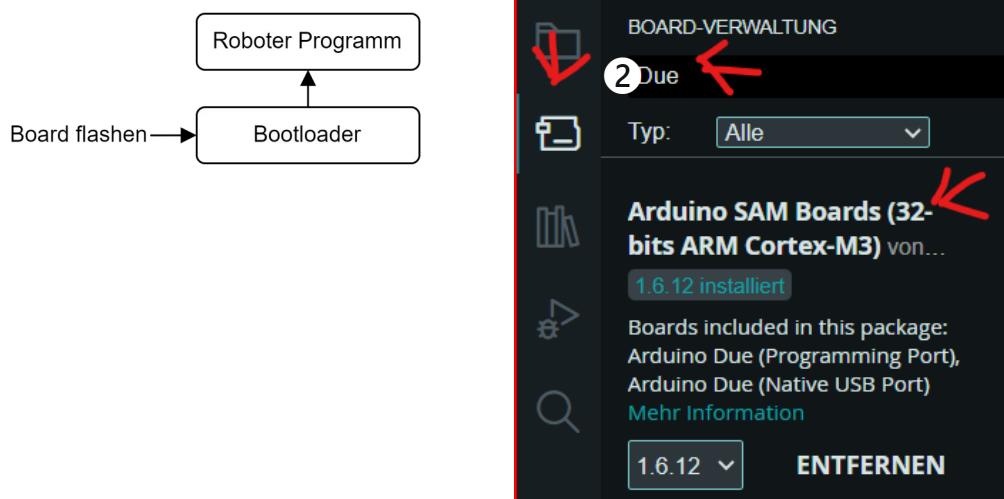


Boards installieren: (Arduino Due und Arduino Nano RP2040 Connect):

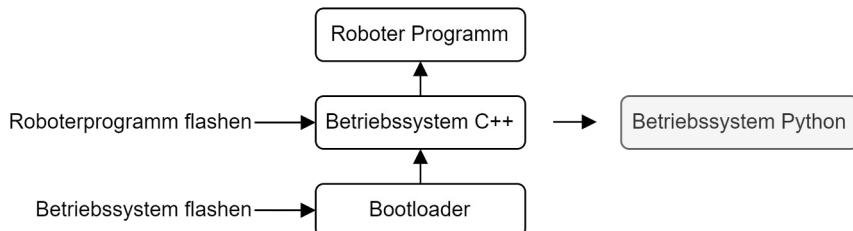
- 1 Zusätzliche Boardverwalter URL eintragen: https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json



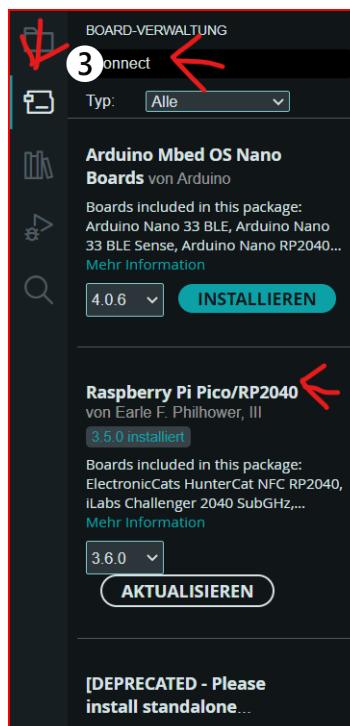
- 2 Arduino Due installieren. Der Due hat zwar einen Bootloader, aber kein Betriebssystem (OS). Das Roboter-Programm läuft also „Bare Metal“:



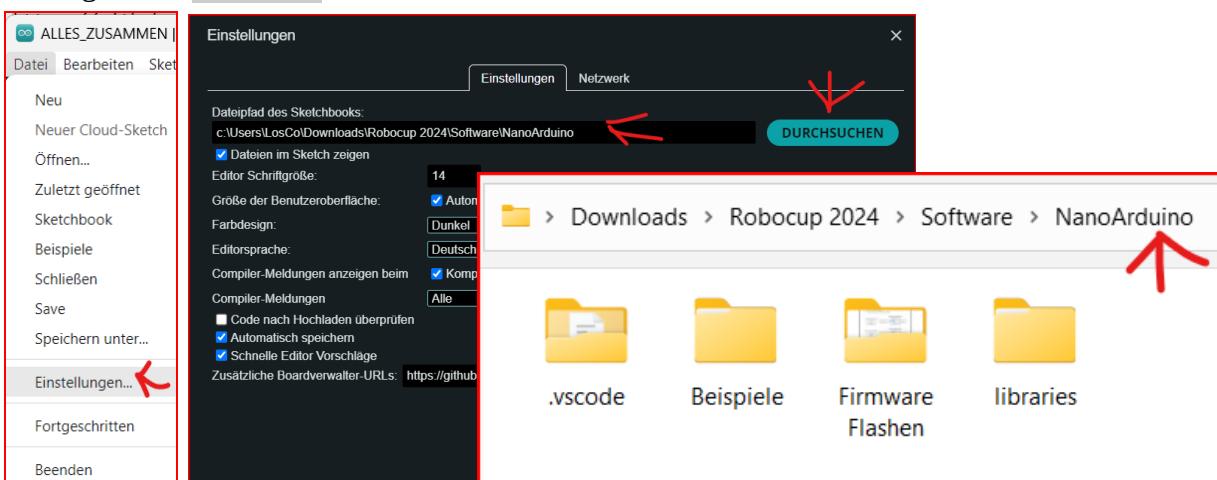
- 3 Arduino Nano RP2040 Connect installieren. Dieser Arduino ist in Wirklichkeit ein kleiner Raspberry Pi und hat somit ein Betriebssystem (OS). Das Betriebssystem kann man austauschen und somit eins für C++ oder Python draufspielen:



Wir benutzen nicht das offizielle OS von Arduino, sondern ein verbessertes für den Raspberry Pi Pico. Nur mit diesem OS bekommt man einen zweiten I2C Bus:

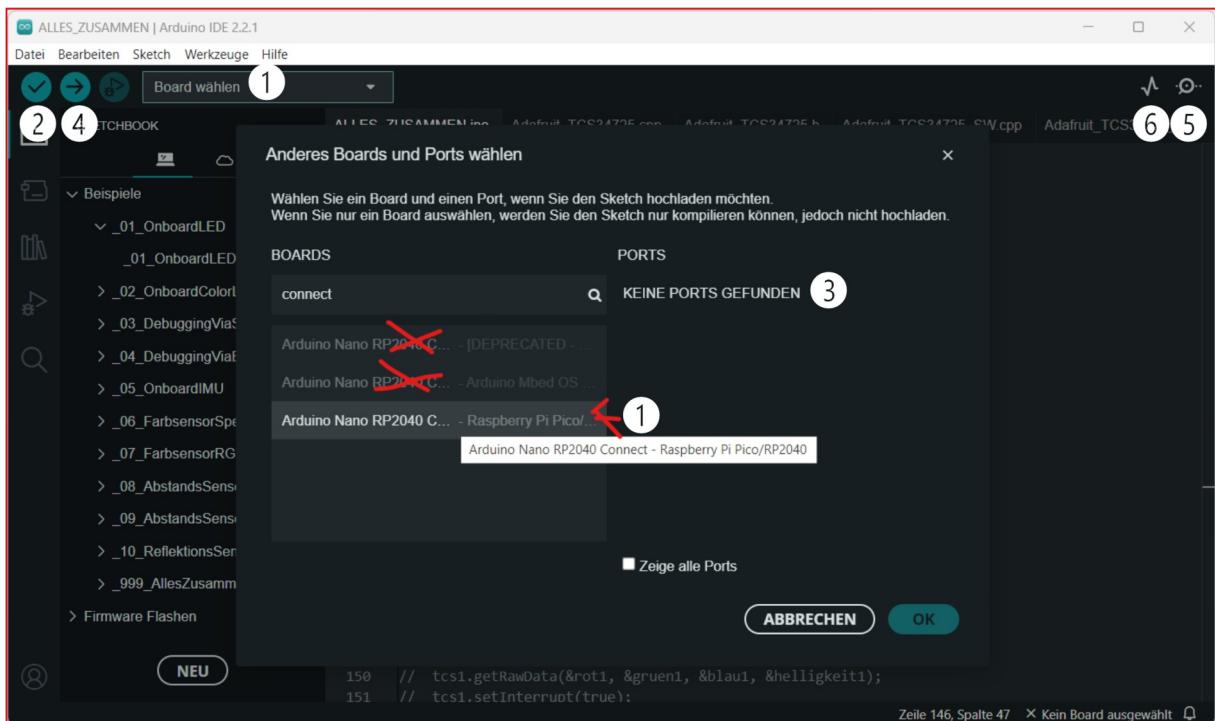


C++ Workspace öffnen: Wir wollen, dass unsere Programme portabel sind, d.h. Libraries sollen nicht in den eigenen Dateien liegen. Die Programme sollen auf einem USB Stick funktionieren und auf den Laptops beim Wettbewerb. Man kann den Ordner konfigurieren, wo sich die Libraries befinden. Im Robotik Tausch liegt ein Beispiel-Workspace namens `ArduinoNano`. Diesen auswählen. Da liegt unser `libraries` Ordner drin.



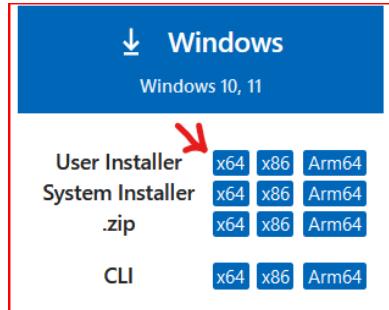
Jetzt kann man alle .ino Dateien des Ordners `ArduinoNano` öffnen (Beispielprogramme oder eigene Programme). Es öffnet sich dafür immer ein neues Arduino Fenster.

- ① Board auswählen (1 mal, bevor man anfängt zu arbeiten)
- ② Projekt bauen
- ③ Port auswählen (jedes Mal, nachdem man den Arduino vom USB getrennt hat)
- ④ Board flashen
- ⑤ Serial Monitor
- ⑥ Serial Plotter



2.1.3 Visual Studio Code mit Arduino Extension

Download und Installation: <https://code.visualstudio.com/Download>



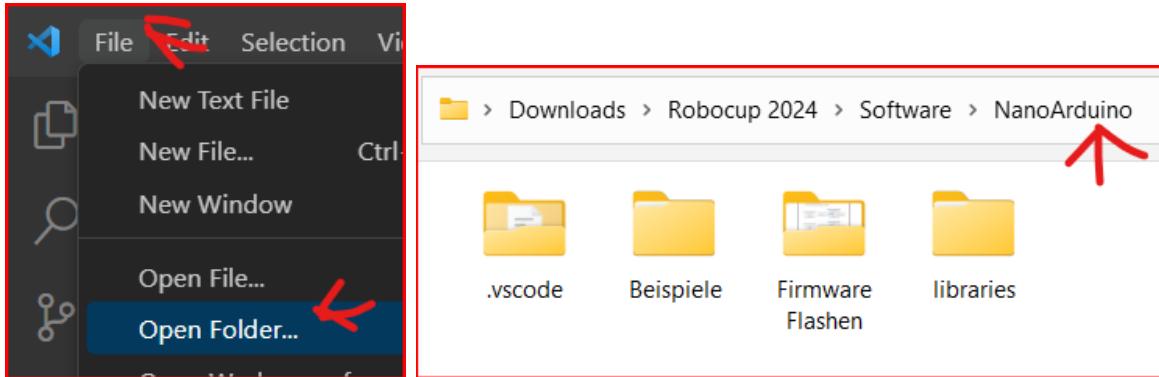
Boards installieren: Muss man mit [2.1.1 Arduino \(1.x\)](#) oder mit [2.1.2 Arduino IDE \(2.x\)](#) machen. Für die folgenden Schritte muss auch jeden Fall [2.1.2 Arduino IDE \(2.x\)](#) installiert sein! Sonst kann man nichts bauen oder flashen!

Arduino Extension installieren und konfigurieren:

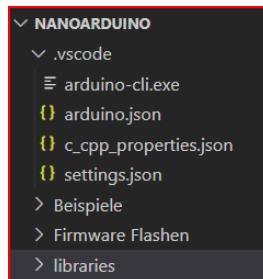
The composite screenshot illustrates the steps to install and configure the Arduino extension in Visual Studio Code. It includes:

- Left Panel:** Shows the 'Programme' (Programs) section with 'Visual Studio Code' highlighted by a red arrow. Below it, the 'Dokumente' (Documents) and 'Dateien' (Files) sections are shown.
- Middle Panel:** Shows the 'EXTENSIONS: MARKETPLACE' search results for 'Arduino'. Several entries are listed, with the first two highlighted by red arrows:
 - 'Arduino' by Microsoft (102ms)
 - 'Arduino' by moozyk (112K)A small red arrow also points to the 'Install' button for the Microsoft entry.
- Right Panel:** Shows the 'Arduino' extension settings in the VS Code sidebar. A red arrow points to the gear icon in the top right corner of the extension card. The settings shown are:
 - Arduino: Clear Output On Build**: A checked checkbox for clearing output logs before uploading or verifying.
 - Arduino: Log Level**: A dropdown menu set to 'verbose'.
 - Arduino: Use Arduino Cli**: A checked checkbox for using the Arduino CLI instead of the legacy version.

C++ Workspace öffnen: Wir wollen, dass unsere Programme portabel sind, d.h. Libraries sollen nicht in den eigenen Dateien liegen. Die Programme sollen auf einem USB Stick funktionieren und auf den Laptops beim Wettbewerb. Man kann den Ordner konfigurieren, wo sich die Libraries befinden. Im Robotik Tausch liegt ein Beispiel-Workspace namens `ArduinoNano`. Diesen auswählen. Da liegt unser `libraries` Ordner drin.

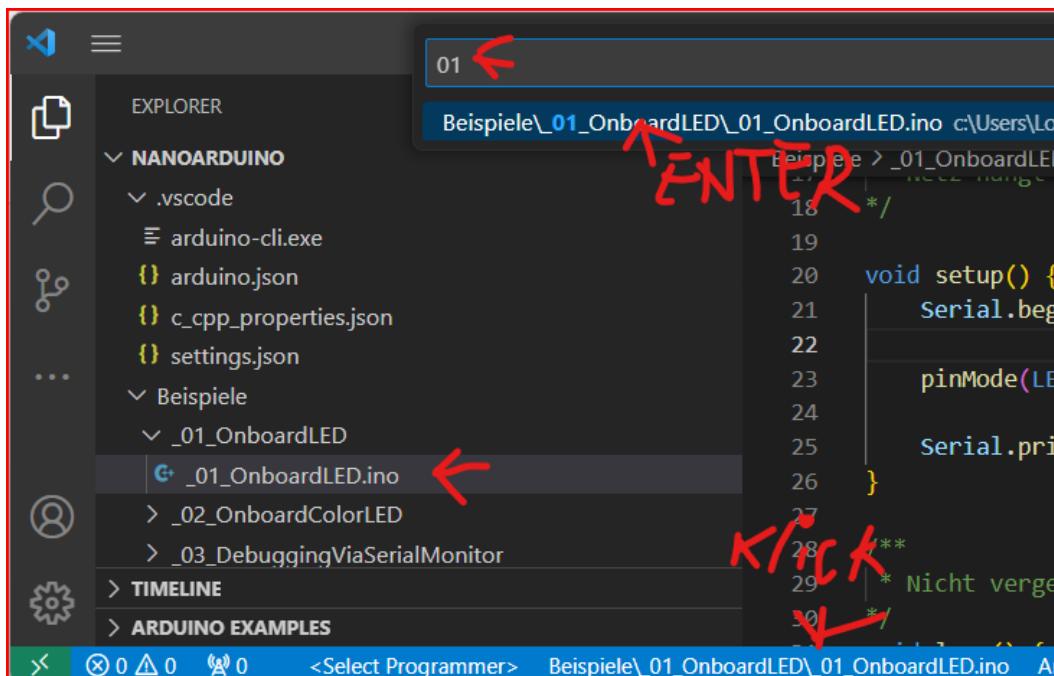


Der `.vscode` Ordner beinhaltet eine ganze Menge Magie, damit Code Completion und Linting funktioniert:

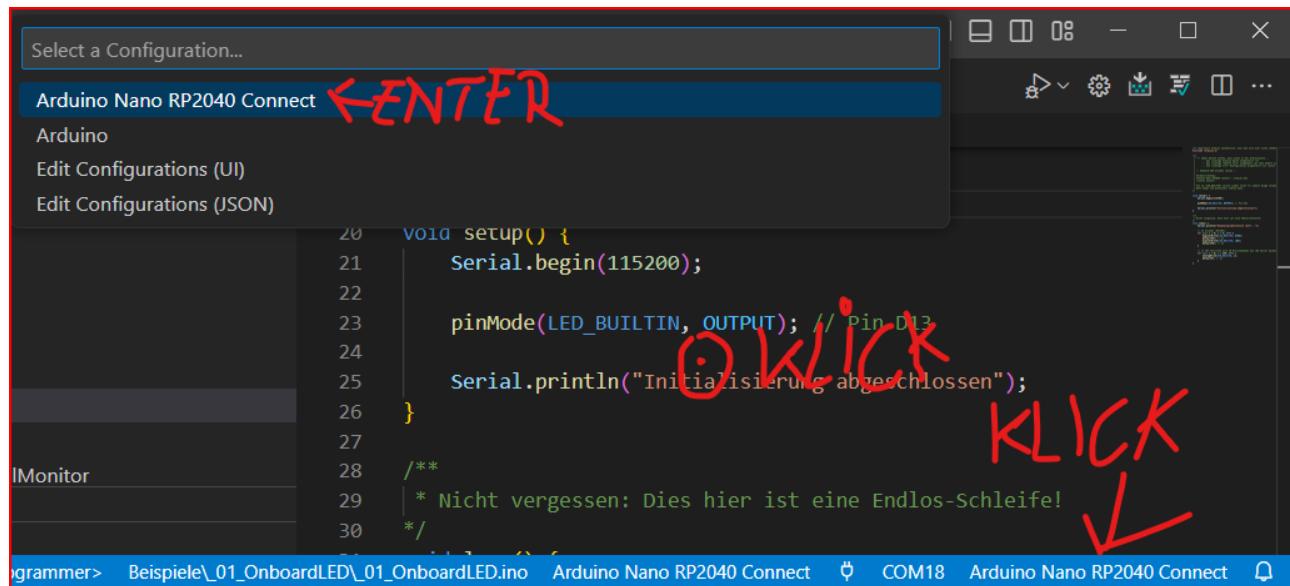


Jetzt kann man alle Dateien des Ordners `ArduinoNano` im selben Fenster öffnen (Beispielprogramme oder eigene Programme).

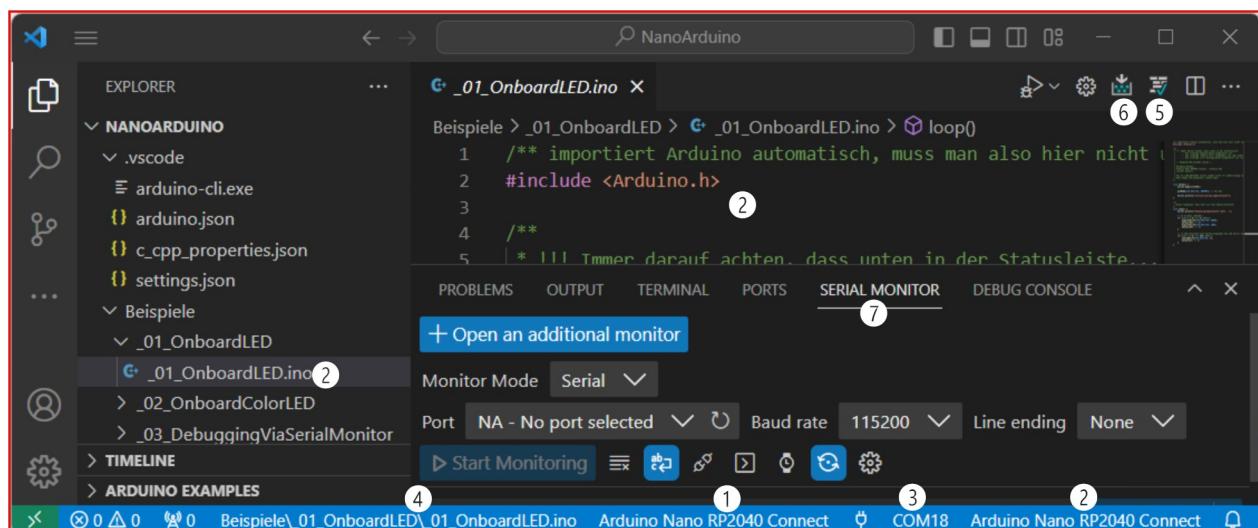
WICHTIG: Der Editor weiß nicht automatisch, welche der geöffneten Programme man bauen oder flashen möchte! Beim Wechsel auf eine andere .ino Datei muss man auch immer manuell umstellen:



WICHTIG: Der Editor weiß nicht automatisch, für welches Board Code Completion und Linting angeboten werden soll. Beim Wechsel auf ein anderes Board muss man das manuell umstellen:



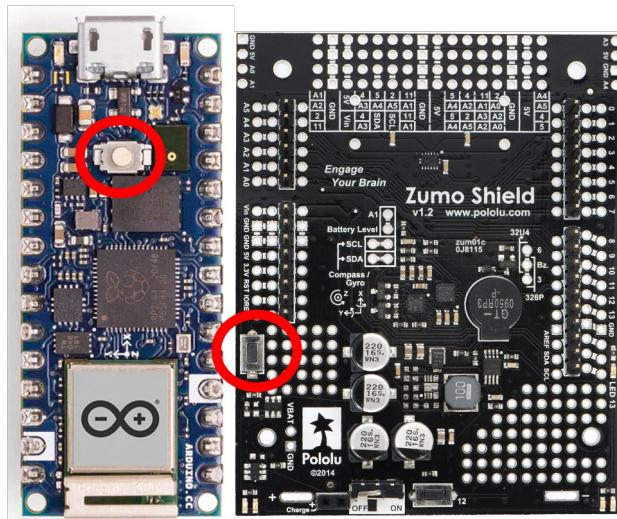
- ① Board auswählen (1 mal, bevor man anfängt zu arbeiten)
- ② Eine `.ino` Datei öffnen, den Cursor hineinsetzen und Code Completion / Linting auswählen (1 mal, bevor man anfängt zu arbeiten)
- ③ Port auswählen (hoffentlich nur 1 mal, bevor man anfängt zu arbeiten)
- ④ Auswählen, welche `.ino` Datei gebaut oder geflasht werden soll (wenn man das Arduino Projekt wechselt)
- ⑤ Projekt bauen
- ⑥ Board flashen
- ⑦ Serial Monitor



2.1.4 Problembehandlung

Manchmal wird der Arduino nicht erkannt und kann nicht geflasht werden oder es funktioniert nach dem Flashen der Serial Monitor nicht.

1. Soft Reset 2 Sekunden gedrückt halten und dann loslassen. Eventuell Port neu auswählen (könnte sich verändert haben). Serial Monitor stoppen und neu starten.



2. Hard Reset: Roboter 2 Sekunden ausschalten oder USB Kabel abziehen, dann Roboter wieder einschalten oder USB Kabel anstecken. Eventuell Port neu auswählen (könnte sich verändert haben). Serial Monitor stoppen und neu starten.
3. Bei Arduino Nano RP2040 Connect: Eventuell ist ein falsches Betriebssystem geflasht (Python, wenn man C++ machen will oder umgekehrt). Selbst wenn das richtige Betriebssystem drauf ist, kann es sein, dass es abgestürzt ist (siehe [3.2.5 Abstürze](#)). Bootmodus umstellen, sodass der Bootloader betreten wird und Betriebssystem neu flashen (siehe [3.2.4 Bootloader laden und Betriebssystem flashen](#)).
4. Es kann auch sein, dass ein Roboter-Programm drauf ist, welches die USB Kommunikation stört (z.B. Motoren wollen Strom, den der USB nicht liefern kann).
 - Arduino Due: Motoren vom Roboter trennen und erneut probieren.
 - Arduino Nano RP2040 Connect: [3.2.4 Bootloader laden und Betriebssystem flashen](#) In diese Modus läuft das Roboter Programm nicht und sollte auch nicht stören.

2.2 Für Programmiersprache Python mit Circuit Python API

TODO Vergleich IDE: Mu, Thonny, Arduino Lab for Circuit Python, VSCode (geht nicht mit Due)

2.2.1 Vorbereitungen

TODO OS flashen

2.2.2 Arduino Lab for Circuit Python

TODO

2.2.3 Visual Studio Code mit Python Extension

TODO .vscode Ordner für Code Completion und Linting erstellen

3 Controller Vergleich

Eigenschaft	<u>3.1 Arduino Due</u>	<u>3.2 Arduino Nano RP2040 Connect</u>	<u>3.3 OpenMV Cam H7</u>
Pins	weiblich (Buchsen)	männlich (Stecker)	weiblich (Buchsen) männlich (Stecker)
einsteckbar auf...	ZumoShield (kopfüber)	Steckbrett	Steckbrett
Anzahl Digital In/Out Pins...	54	20	10
...davon Anzahl PWM Pins	12	18	10 (3 Servo-fähig)
...davon Anzahl Analog In Pins	12	6	1
...davon Anzahl I2C Bus	2x I2C → 4 Pins	2x I2C → 4 Pins	2x I2C → 4 Pins
...davon Anzahl SPI Bus	4x SPI → 12 Pins	1x SPI → 3 Pins	1x SPI → 3 Pins
...davon Anzahl UART	3x UART → 6 Pins	1x UART → 2 Pins	2x UART → 4 Pins
Batterie-Spannung	7..12 V	5..21V	3,6..5V → benötigt Spannungsregler
Max Strom 5V Pin	800 mA	X nicht verfügbar	X nicht verfügbar
Max Strom 3V3 Pin	800 mA	800 mA	250 mA
Logik-Spannung	3,3 V	3,3 V	3,3 V
Ausgangs-Strom pro Pin	3 mA	4 mA	12 mA
Breite	53.3 mm	18 mm	36 mm
Länge	101.5 mm	45 mm	45 mm
CPU-Takt	84 MHz	133 MHz	480 MHz
RAM	96 kB	264 kB	832 kB
Programmiersprache C++	✓	✓	X
Programmiersprache Python	X	✓	✓
Onboard LED	✓	✓	X

Onboard RGB LED	X	<input checked="" type="checkbox"/> (nur C++, nicht, wenn Bluetooth aktiv ist)	<input checked="" type="checkbox"/>
Onboard 5V Regler	<input checked="" type="checkbox"/> (800 mA)	X	X
Onboard 3,3V Regler	<input checked="" type="checkbox"/> (800 mA)	<input checked="" type="checkbox"/> (800 mA)	<input checked="" type="checkbox"/> (250 mA)
Onboard Soft Reset	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	X
Onboard Hard Reset	<input checked="" type="checkbox"/> (mit ZumoShield)	X	X
Onboard 2-Kanal DC-Motortreiber	<input checked="" type="checkbox"/> (mit ZumoShield)	X	X
Onboard Summer	<input checked="" type="checkbox"/> (mit ZumoShield)	X	X
Onboard 6-Achsen IMU	<input checked="" type="checkbox"/> (mit ZumoShield)	<input checked="" type="checkbox"/>	X
Onboard Temperatursensor	X	<input checked="" type="checkbox"/>	X
Onboard Mikrofon	X	<input checked="" type="checkbox"/>	X
Onboard WiFi	X	<input checked="" type="checkbox"/>	X
Onboard Bluetooth	X	<input checked="" type="checkbox"/>	X
Onboard Kamera	X	X	<input checked="" type="checkbox"/>

3.1 Arduino Due

Produkt-Link: <https://store.arduino.cc/products/arduino-due>

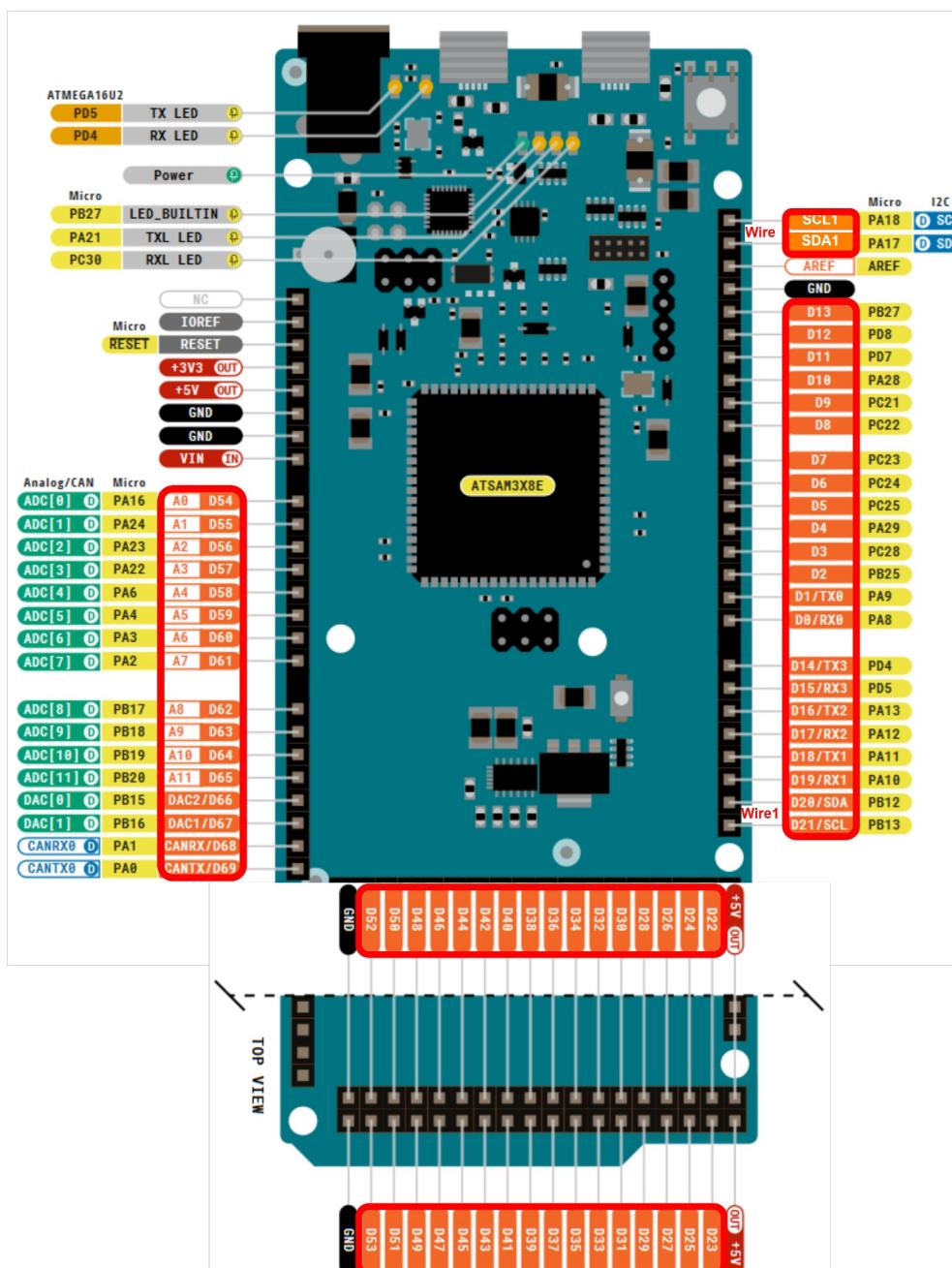
Dieser Controller kann nur mit „Arduino C++“ (siehe [2.1 Für Programmiersprache C++ mit Arduino API](#)) programmiert werden. Er hat wesentlich mehr Pins als der Nano Connect, ist dadurch aber sperriger. So richtig mithalten kann der Due eigentlich nur, weil man ihn direkt auf das ZumoShield stecken kann, welches einen Akku-Anschluss, Hard Reset Schalter, 2-Kanal Motortreiber und 9-Achsen Bewegungssensor mitbringt.

Weitere Details: siehe [3 Controller Vergleich](#)

3.1.1 Beispiel-Programme

TODO

3.1.2 Pinout



Die rot eingeklammerten Bezeichnungen kann man im Programm verwenden.

Beispiel C++ digitaler Pin D13: Im Programm als 13 ansprechen:

```
#include <Arduino.h>
void setup() {
    pinMode(13, OUTPUT);
}
```

Beispiel C++ analoger Pin A7: Im Programm als A7 ansprechen:

```
#include <Arduino.h>
void setup() {
    pinMode(A7, OUTPUT);
}
```

Beispiel C++ I2C Pins SDA1/SCL1: Im Programm als Wire ansprechen:

```
#include <Arduino.h>
#include <Wire.h>

void setup() {
    Wire.begin();
}
```

- SCL1/SDA1 sind I2C0 (Wire)
- D20/D21 sind I2C1 (Wire1)

3.2 Arduino Nano RP2040 Connect

Produkt-Link: <https://store.arduino.cc/products/arduino-nano-rp2040-connect-with-headers>

Dieser Controller kann sowohl mit „Arduino C++“ als auch MicroPython programmiert werden (momentan haben wir aber nur Beispiel-Programme für C++). Er steht dem Arduino Due in nichts nach, außer, dass er viel weniger Pins hat. Da wir abgesehen von den Motoren aber eh alles an I2C Busse anschließen, ist dies kein großer Nachteil. Onboard befinden sich eine RGB LED, ein 6-Achsen Bewegungssensor und ein Bluetooth-Modul. Bluetooth ist im Wettbewerb untersagt, aber sehr praktisch fürs testen/debuggen.

Weitere Details: siehe [3 Controller Vergleich](#)

3.2.1 Beispiel-Programme

Beispiel-Programm Onboard LED:

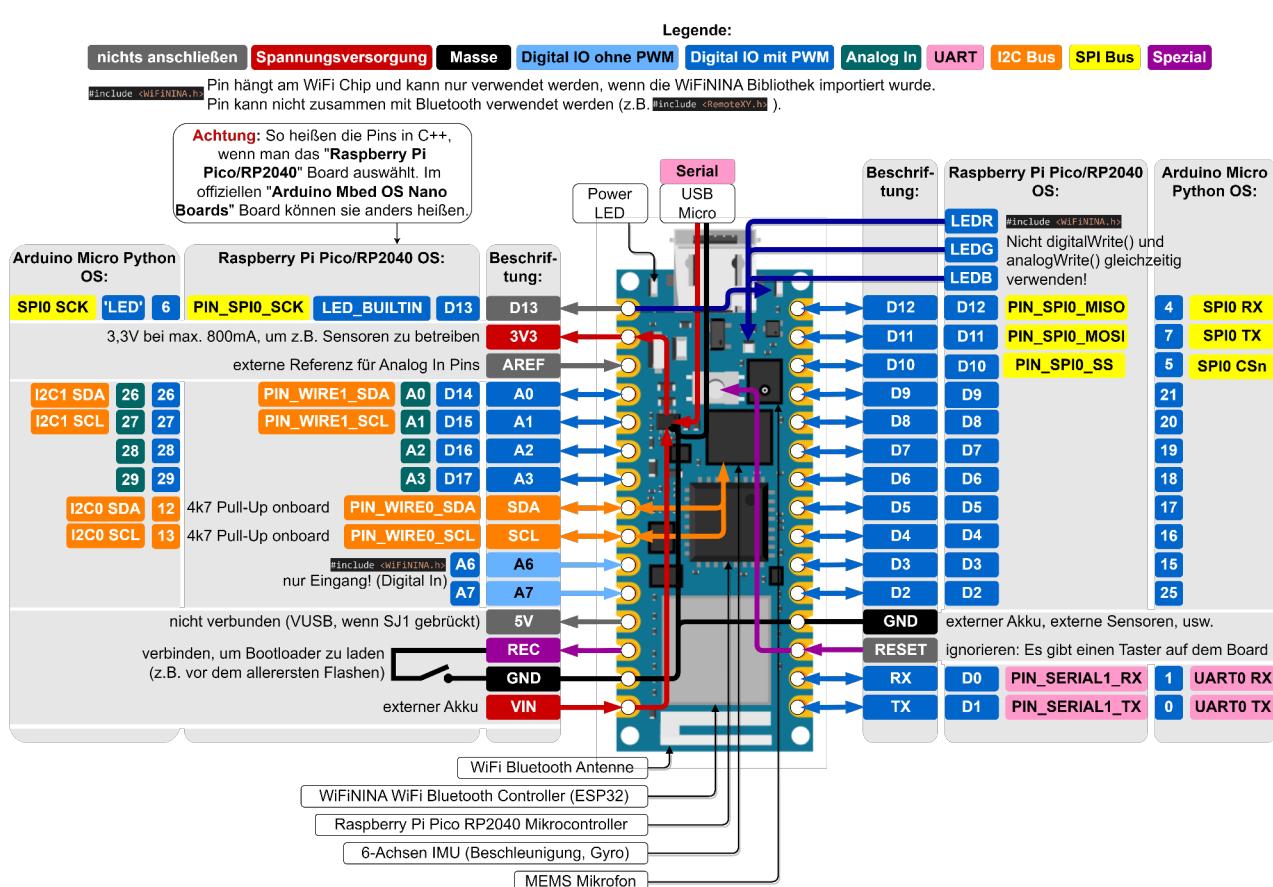
Beispiele\01_OnboardLED\01_OnboardLED.ino

Beispiel-Programm Onboard RGB LED:

Beispiele\02_OnboardColorLED\02_OnboardColorLED.ino

Beispiel-Programm Onboard 6-Achsen Bewegung

3.2.2 Pinout



Beispiel C++ digitaler Pin D13: Im Programm als D13 oder LED_BUILTIN ansprechen:

```
#include <Arduino.h>
void setup() {
    pinMode(D13, OUTPUT);
}
```

Beispiel Python digitaler Pin D13: Im Programm als 6, 'D13' oder 'LED' ansprechen:

```
from machine import Pin
led = Pin(6, Pin.OUT)
#led = Pin('D13', Pin.OUT)
#led = Pin('LED', Pin.OUT)
```

Beispiel C++ analoger Pin A7: Im Programm als A7 oder ansprechen:

```
#include <Arduino.h>
void setup() {
    pinMode(A7, OUTPUT);
}
```

Beispiel Python analoger Pin A3: Im Programm als 29 oder 'A3' ansprechen:

```
from machine import Pin
led = Pin(29, Pin.OUT)
#led = Pin('A3', Pin.OUT)
```

Beispiel C++ I2C Pins PIN_WIRE0_SDA/PIN_WIRE0_SCL: Im Programm als wire ansprechen:

```
#include <Arduino.h>
#include <Wire.h>

void setup() {
    Wire.begin();
}
```

Beispiel Python I2C Bus 0 mit Pins SDA/SCL:

```
from machine import Pin, I2C
derErsteBus = I2C(0, scl=Pin(13), sda=Pin(12), freq=100_000)
```

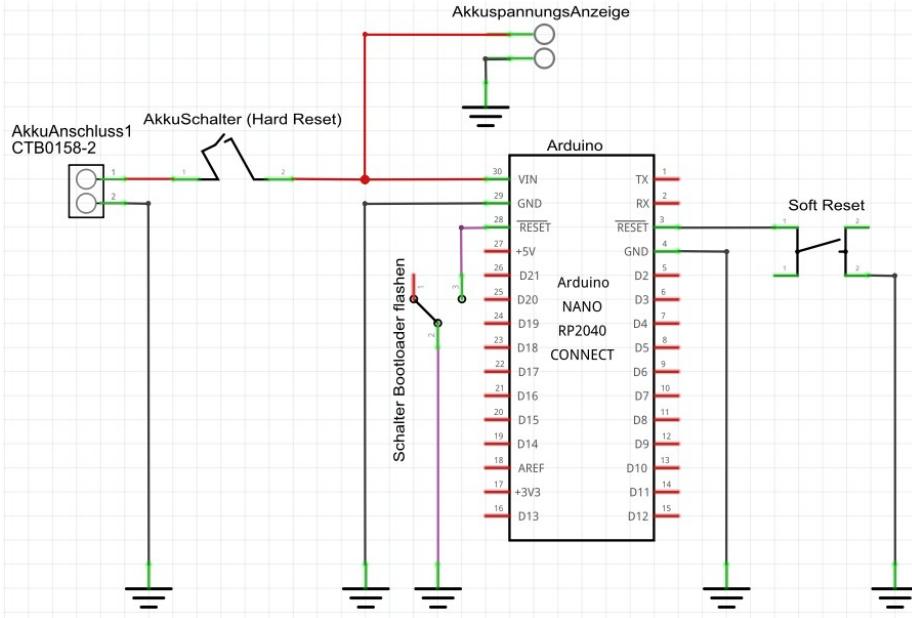
- SCL/SDA sind I2C0 (Wire)
- A0/A1 sind I2C1 (Wire1)

3.2.3 Schaltplan

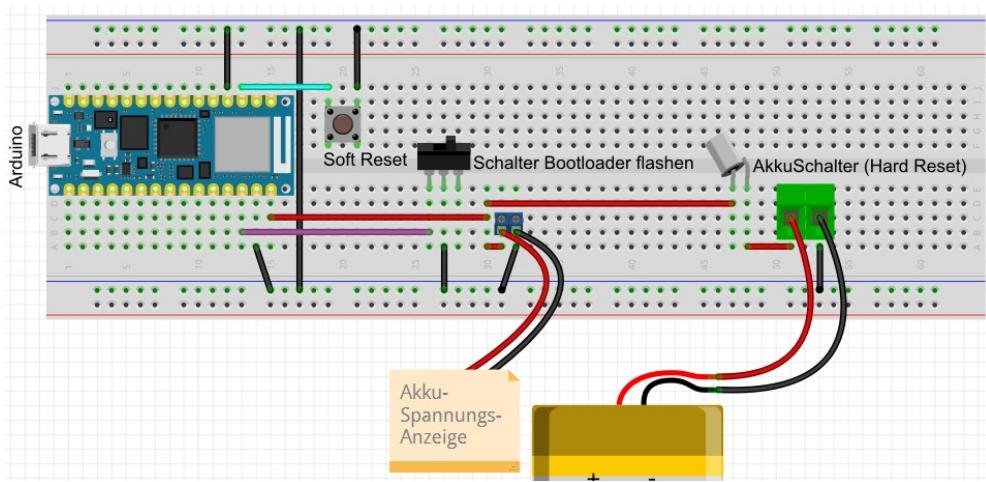
Schaltplan: Akku-Anschluss + Schalter (Hard Reset) + Akku-Spannungsanzeige + Arduino + Taster (Soft Reset) + Schalter (Bootloader flashen)

- Der Akku wird an die Schraubklemme „AkkuAnschluss“ angeschlossen.

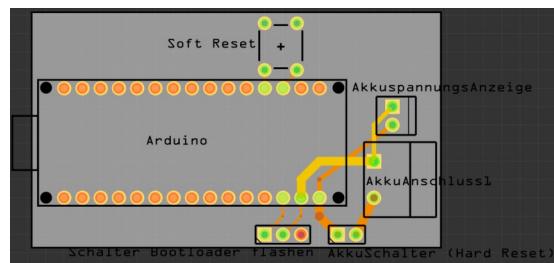
- Dann kommt der „AkkuSchalter“, mit dem man den ganzen Roboter ein- und ausschalten kann, ohne den Akku-Stecker abziehen zu müssen.
- Dann wird verteilt auf die Akku-Spannungsanzeige und den Arduino.
- Ein Taster kann gedrückt werden, um den Arduino neu zu starten (Soft Reset).
- Ein Schiebeschalter stellt ein, ob man den Bootloader laden will (z.B. um das Betriebssystem neu zu flashen) oder das Roboter-Programm starten will.



So sieht es z.B. auf einem Steckbrett aus:



So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



3.2.4 Bootloader laden und Betriebssystem flashen

TODO

3.2.5 Abstürze

TODO LED 4x lang, 4x kurz

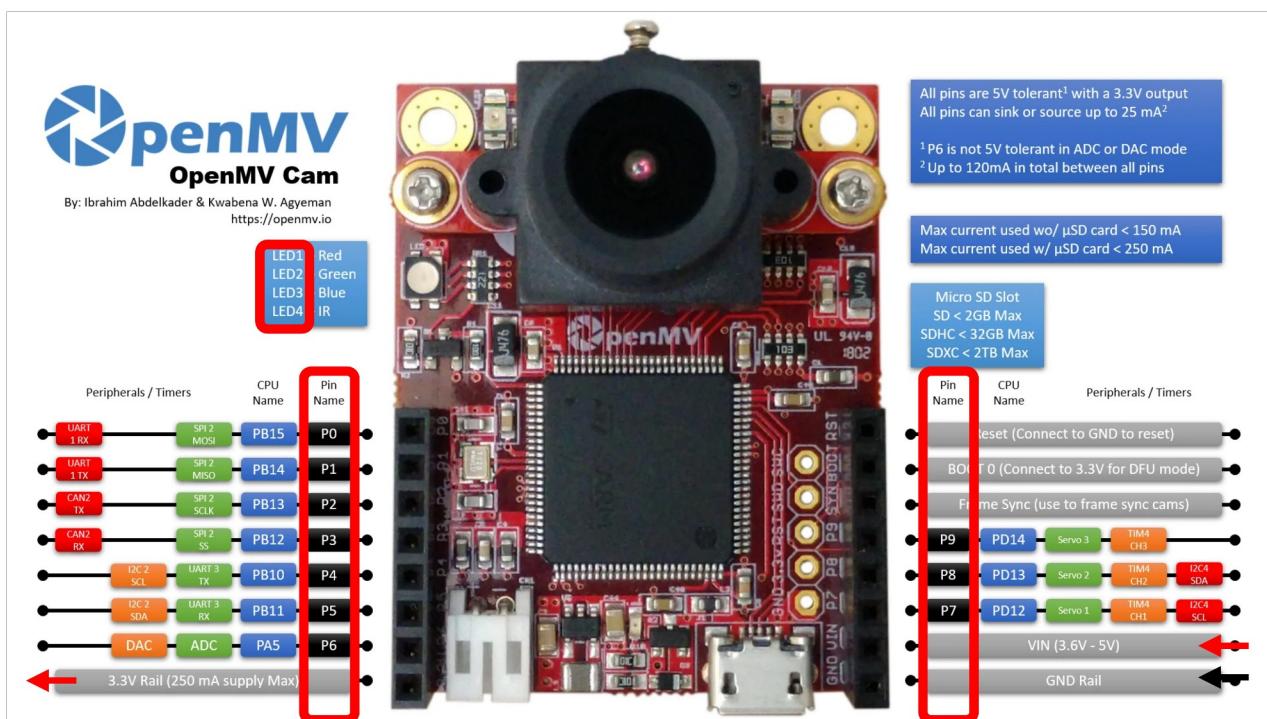
3.3 OpenMV Cam H7

Dieser Controller kann ausschließlich mit MicroPython programmiert werden und ist auf die Verarbeitung von Kamera-Bildern spezialisiert. Da die OpenMV Cam nur sehr wenige Pins hat und wir momentan nur Bibliotheken und Beispielprogramme für den Arduino haben, kann man die OpenMV Cam so programmieren als wäre sie ein Sensor und an einen Arduino (Haupt-Controller) anschließen, z.B. über UART, I2C oder SPI ([siehe diese Arduino Bibliothek](#)). Der Haupt-Controller lässt dann das eigentliche Roboter-Programm laufen und holt sich vom „Kamera-Sensor“ Informationen über die erkannten Objekte. Man muss also 2 Programme schreiben.

- Produkt-Link: <https://openmv.io/collections/cams/products/openmv-cam-h7>
- Video Linien-Erkennung: <https://www.youtube.com/watch?v=oXexWgi3ldg>
- Video Farb-Erkennung: <https://www.youtube.com/watch?v=h9VoDcROkxc>
- Video Kreis-Erkennung: <https://www.youtube.com/watch?v=1M6uOoHYN9o>
- Video Rechteck-Erkennung: https://www.youtube.com/watch?v=zRoprce5W_Y

Weitere Details: siehe [3 Controller Vergleich](#)

Pinout:



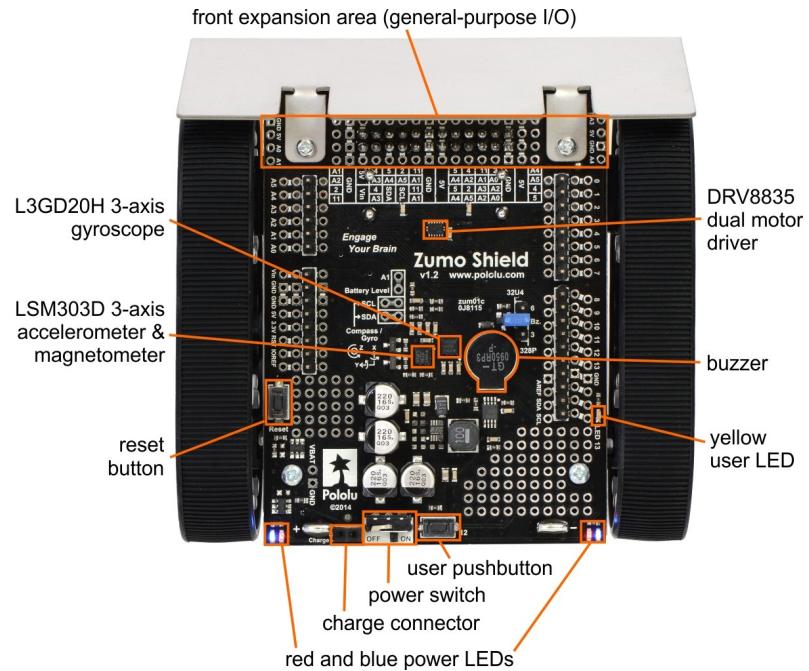
Die rot eingerahmten Bezeichnungen werden im Programm verwendet.

TODo Python Beispiele

4 Bauteile

4.1 ZumoShield

Produkt-Link: <https://www.pololu.com/product/2508>



4.2 DC-Motoren und Servos

4.2.1 2-Kanal DC-Motorsteuerung

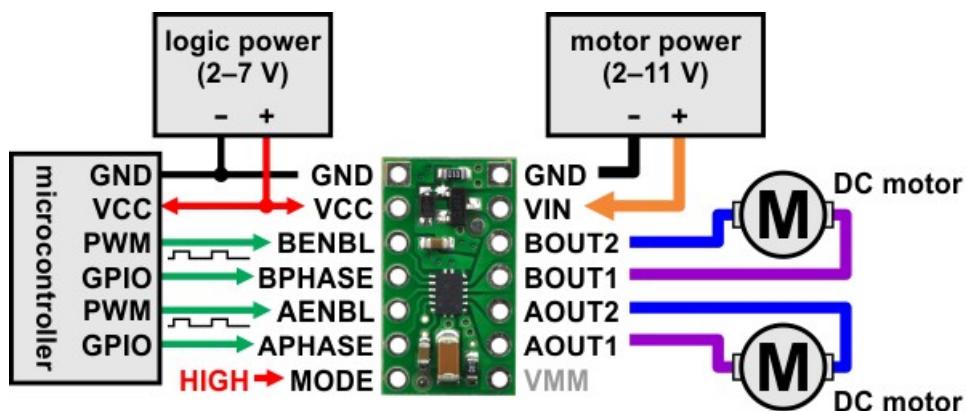
Produkt-Link: <https://www.pololu.com/product/2135>

Beispiel-Programm: TODO

Eine Motorsteuerung wird benötigt, um die Steuersignale des Controllers für die Motoren zu verstärken. Die Motoren benötigen sowohl mehr Strom als auch mehr Spannung, als der Arduino liefern kann. Die DRV8835 Motorsteuerung hat 2 Kanäle, kann also 2 Motoren steuern.

Eigenschaft	Wert	OK?
Logik-Spannung	2..7 V	<input checked="" type="checkbox"/> 3,3 V vom Arduino
Motor-Spannung	0..11 V	(a) <input checked="" type="checkbox"/> 7,4 V vom 4.7.1 LiPo Akku (7,4V) : Die Motoren drehen schneller, aber die Geschwindigkeit ändert sich mit dem Ladezustand des Akkus. Die Motoren sind nur für 6V ausgelegt, aber das ist OK: Dann erwärmen sie sich ein wenig. (b) <input checked="" type="checkbox"/> 5 V vom 4.7.5 5V-Spannungsregler (Extern) oder 3.1 Arduino Due : Die Motoren drehen langsamer, aber die Geschwindigkeit ist immer gleich. Für die Motoren OK.
max. Strom pro Kanal	1,2 A	(a) 1 Motor pro Kanal: <input checked="" type="checkbox"/> 0,1A (Leerlauf) .. 1,6A (blockiert) (Pololu 6V HP). Die Motoren sind selten komplett blockiert, also OK. Die Motorsteuerung erwärmt sie sich ein wenig. (b) 2 Motoren pro Kanal: <input checked="" type="checkbox"/> 0,2A (Leerlauf) .. 3,2A (blockiert) (Pololu 6V HP). Die Chance ist groß, dass beim normalen Fahren die 1,2A länger überschritten werden und die Motorsteuerung heiß wird, dass sie durchbrennt.

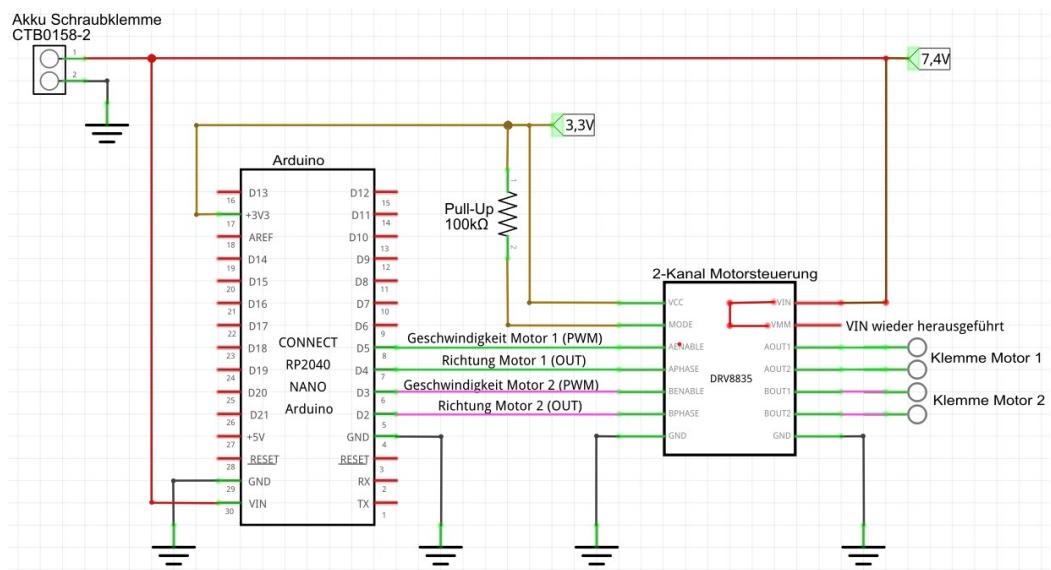
Pinout:



Motorsteuerung Pin	verbinden mit
MODE	Arduino 3,3V (HIGH) via 100k Pull-Up Widerstand
• <u>LOW</u> : „in-in“ Modus (wollen wir nicht verwenden)	
• <u>HIGH</u> : „phase-enable“ Modus (xENBL, xPHASE)	
AENBL – Motor A Geschwindigkeit Eingang	irgendein Arduino PWM Pin
APHASE – Motor A Dreh-Richtung Eingang	irgendein Arduino Digital Out Pin
AOUT1 – Motor A Ausgang 1	Motor A , Anschluss 1
AOUT2 – Motor A Ausgang 2	Motor A , Anschluss 2
BENBL – Motor B Geschwindigkeit Eingang	irgendein Arduino PWM Pin
BPHASE – Motor B Dreh-Richtung Eingang	irgendein Arduino Digital Out Pin
BOUT1 – Motor B Ausgang 1	Motor B , Anschluss 1
BOUT2 – Motor B Ausgang 2	Motor B , Anschluss 2
VCC – Logik-Spannung Eingang	Arduino 3,3V (HIGH)
VIN – Motor-Spannung Eingang	Akku \oplus oder 5V Spannungsregler VOUT
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND
VMM – Motor-Spannung Ausgang	nichts (n.c.)

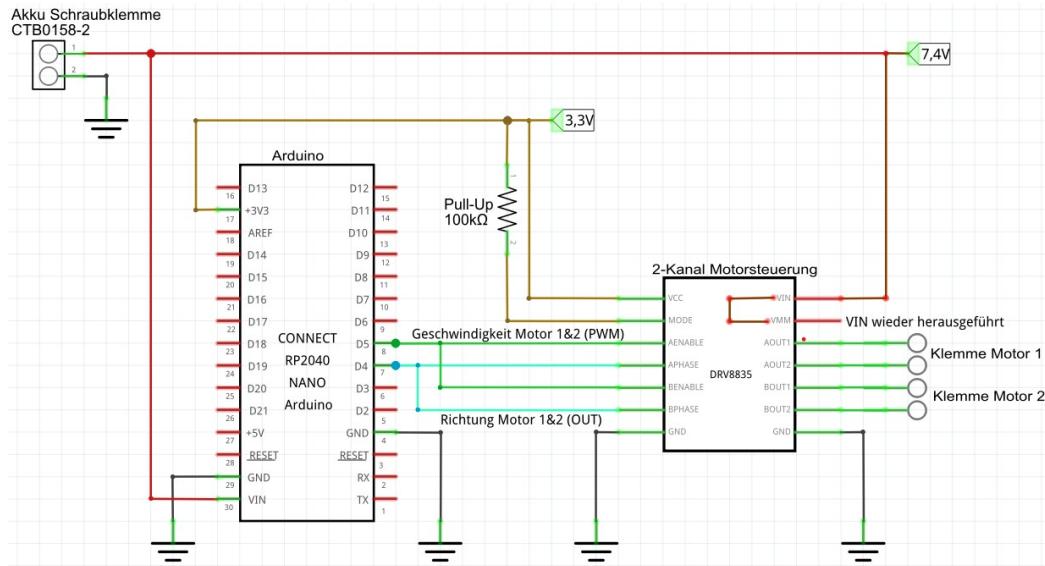
Variante (a): Akku-Anschluss + Arduino (4 Pins verwendet) + Motorsteuerung (VIN: Akku \oplus)

- MODE ist über einen Pull-Up Widerstand mit 3,3V verbunden, um das Signal auf HIGH zu zwingen („phase-enable“ Modus).
- Die Motoren können jeweils einzeln gesteuert werden, also mit unterschiedlichen Geschwindigkeiten und Richtungen.
- Die Motoren werden direkt aus dem Akku versorgt.



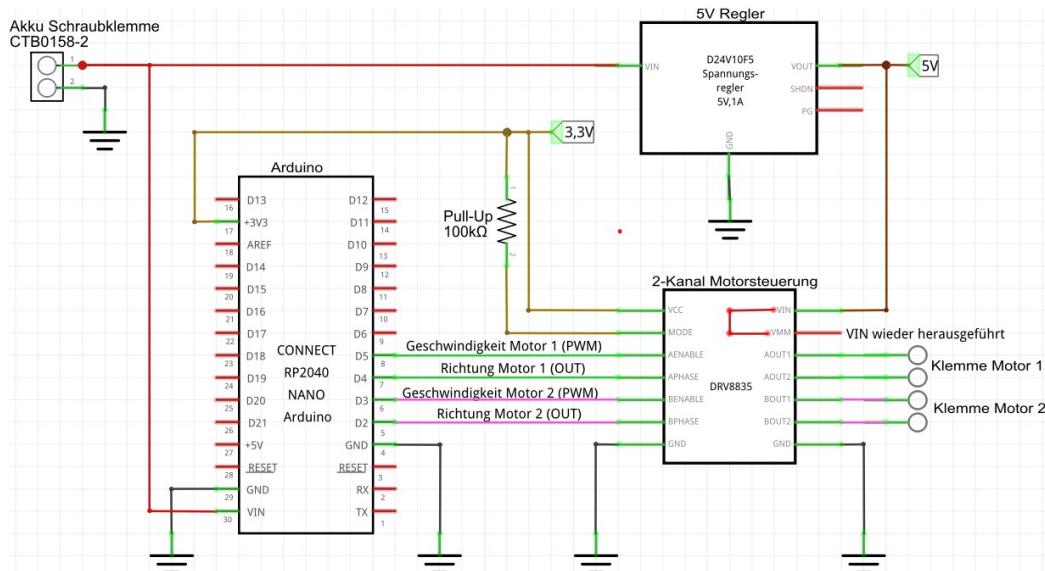
Variante (b): Akku-Anschluss + Arduino (2 Pins verwendet) + Motorsteuerung (VIN: Akku \oplus)

- MODE ist über einen Pull-Up Widerstand mit 3,3V verbunden, um das Signal auf HIGH zu zwingen („phase-enable“ Modus).
 - Wenn man 2 Pins am Arduino einparen möchte (oder muss), kann man einfach beide Motorsteuer-Kanäle miteinander verbinden.
 - Die Motoren werden direkt aus dem Akku versorgt.

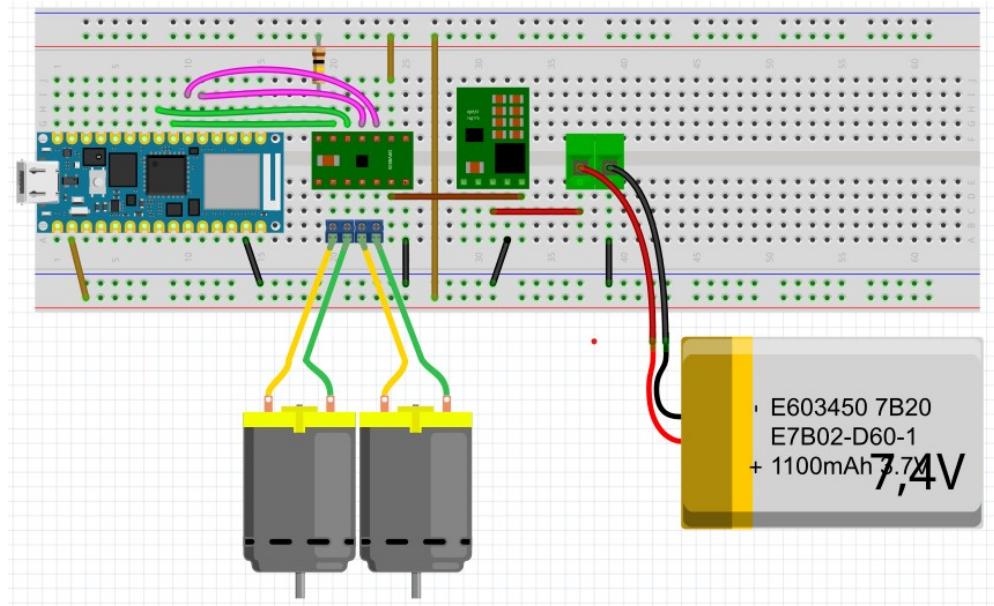


Variante (c): Akku-Anschluss + Arduino (4 Pins verwendet) + Motorsteuerung (VIN: 5V)

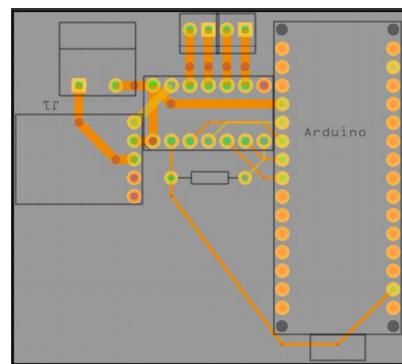
- MODE ist über einen Pull-Up Widerstand mit 3,3V verbunden, um das Signal auf HIGH zu zwingen („phase-enable“ Modus).
 - Die Motoren können jeweils einzeln gesteuert werden, also mit unterschiedlichen Geschwindigkeiten und Richtungen.
 - Die Motoren werden aus einem 5V Regler versorgt.



So sieht Variante (c) z.B. auf einem Steckbrett aus:



So sieht Variante (c) z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



4.2.2 2-Kanal I2C DC-Motorsteuerung

Produkt-Link: <https://learn.sparkfun.com/tutorials/serial-controlled-motor-driver-hookup-guide/introduction>

TODO

4.2.3 DC Motor

Produkt-Link: <https://www.pololu.com/category/60/micro-metal-gearmotors> (Wir benutzen 6V, High-Power. Getriebe-Übersetzung sucht ihr euch aus.)

Beispiel-Programm: siehe [4.2.1 2-Kanal DC-Motorsteuerung](#)

Eigenschaft	Wert	OK?
Motor-Spannung	6V	siehe 4.2.1 2-Kanal DC-Motorsteuerung
Leerlauf-Strom	0,1A	
Blockade-Strom	1,6A	

max. Geschwindigkeit	Hängt von der Getriebe-Übersetzung ab, die ihr euch aussuchen dürft. Siehe Produkt-Link . Wir benutzen für Ketten meist 1:75 (schneller, aber weniger Drehmoment), für Räder 1:250 (langsamer, aber mehr Drehmoment).
max. Drehmoment	

Pinout:



Ein DC-Motor hat 2 Anschlüsse: \oplus und \ominus . Die Spannung zwischen den Pins bestimmt die Geschwindigkeit, der Strom das Drehmoment. Vertauscht man \oplus und \ominus , ändert sich die Drehrichtung.

Schaltplan: siehe [4.2.1 2-Kanal DC-Motorsteuerung](#)

4.2.4 Servo Motor

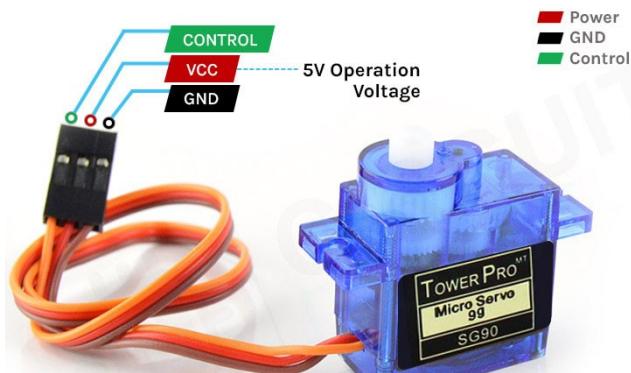
Produkt-Link: beispielsweise <https://www.pololu.com/product/2818>

Beispiel-Programm: [TODO](#)

Ein Servo-Motor ist ein motorisierter Hebel: Die Position des Hebels kann eingestellt werden, aber nicht wie schnell der Servo diese anfährt. Ein Servo-Motor kann nicht im Kreis drehen, sondern meist nur 90° oder 180° . Manche Spezial-Servos können komplett im Kreis drehen („Continuous Rotation Servo“). Es gibt analoge und digitale Servos. Ein Servo hat die Motorsteuerung bereits dabei.

Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> 3,3 V vom Arduino
Motor-Spannung	5..6V	(a) X 7,4 V vom 4.7.1 LiPo Akku (7,4V) : Das macht die Elektronik des Servos kaputt. (b) <input checked="" type="checkbox"/> 5 V vom 4.7.5 5V-Spannungsregler (Extern) oder 3.1 Arduino Due
max. Strom	0,8 A	0,06A (Leerlauf) .. 0,8A (blockiert)

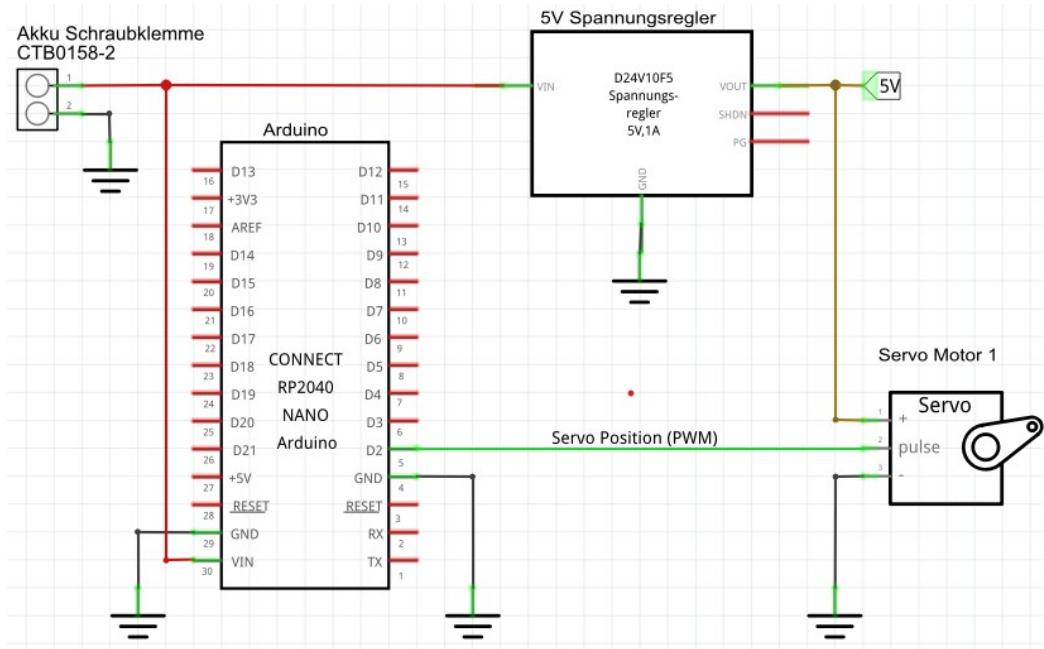
Pinout:



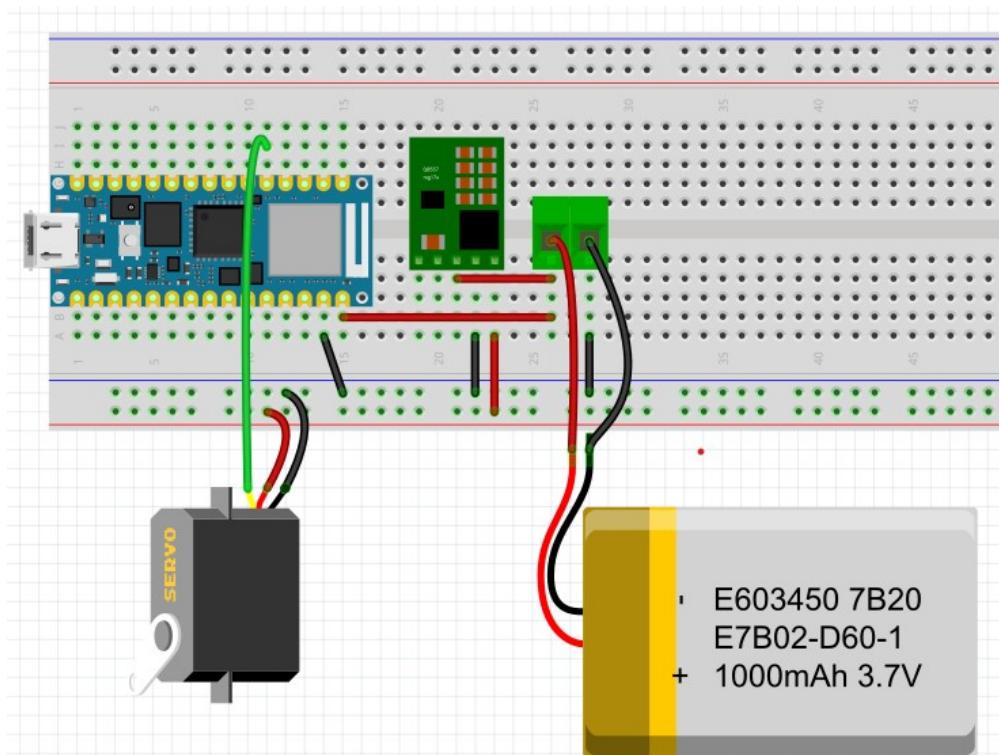
Servo Pin	verbinden mit
CONTROL – Position	irgendein Arduino PWM Pin
VCC – Motor-Spannung Eingang	5V Spannungsregler VOUT
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND

Schaltplan: Akku-Anschluss + Arduino + Servo (VCC: 5V)

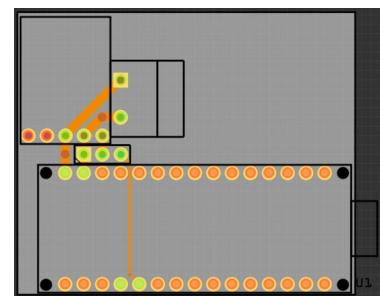
- Der Servo-Motor beinhaltet bereits die Motorsteuerung, muss aber aus einem 6V-Spannungsregler versorgt werden, da die Akku-Spannung zu hoch ist.



So sieht es z.B. auf einem Steckbrett aus:



So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



4.3 User Interface

4.3.1 Programmierbare RGB LED (extern)

TODO

4.3.2 I2C Displays

TODO

4.4 Optische Sensoren

4.4.1 OpenMV Cam

TODO

4.4.2 RGB Farbsensor

Produkt-Link: <https://www.adafruit.com/product/1334>

Beispiel-Programm: [Beispiele_07_FarbsensorRGB_07_FarbsensorRGB.ino](#)

Der RGB-Farbsensor leuchtet die Umgebung mit weißem Licht aus und misst die Intensität des reflektierten Lichts. Eine Möglichkeit, Farben zu mischen ist mit [rot-, blau- und grün-Anteilen \(RGB\)](#). Da die RGB Farbanteile auch die Helligkeit beinhalten, ist es nicht so einfach eine Farbe wie grün oder rot sauber voneinander zu unterscheiden. Dafür kann man den zusätzlichen Helligkeits-Kanal verwenden oder die 3 RGB Kanäle in das HSV-Farbsystem umrechnen.

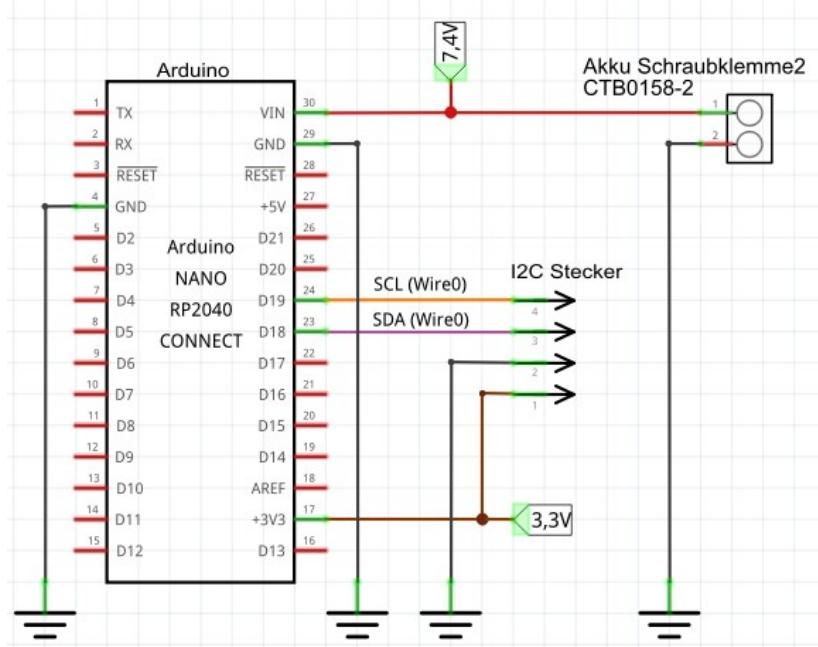
Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> 3,3 V vom Arduino
Stromaufnahme	0,038 A	siehe 1.8 Stromaufnahme
I2C Adresse	0x29	siehe 1.6 I2C Adressen

Pinout:

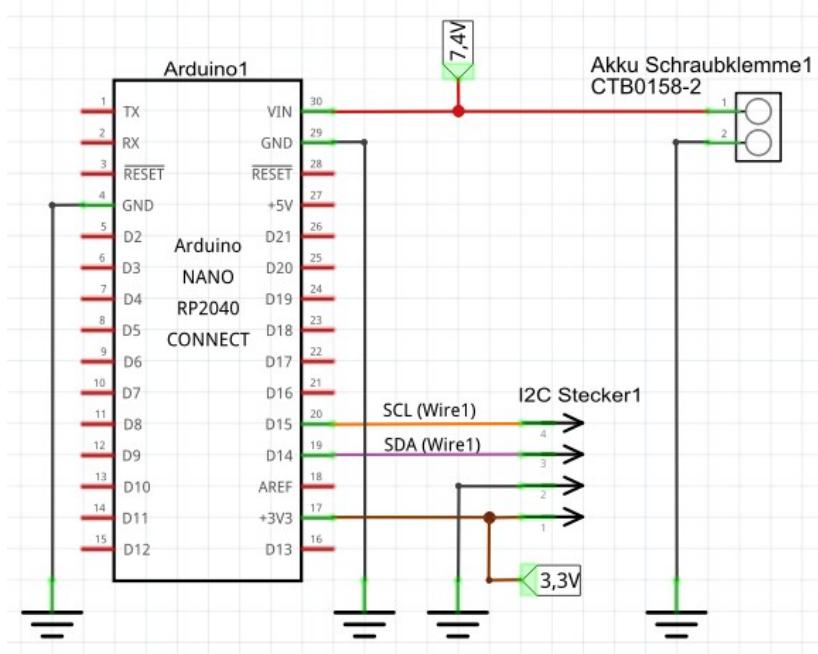


TCS34725 Pin	verbinden mit
SDA – I2C Bus Daten Eingang/Ausgang	Arduino SDA
SCL – I2C Bus Takt Eingang	Arduino SCL
VIN – Logik-Spannung Eingang	Arduino 3,3V
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND
3V3 – 3,3V Ausgang	nichts (n.c.)
LED – kann man optional an irgendeinen Digital-Pin oder den INT Pin vom Sensor anschließen, um die LED abzuschalten	nichts (n.c.)
INT – kann man optional an irgendeinen Digital-Pin anschließen, wenn man den Sensor effizienter auslesen will	nichts (n.c.)

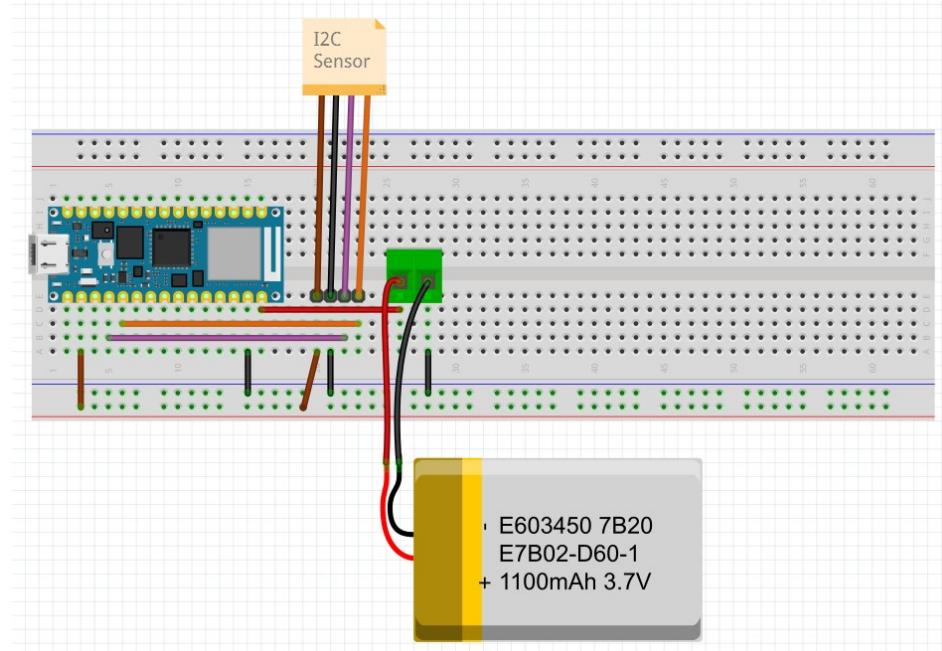
Variante (a): Akku-Anschluss + Arduino + TCS34725 angeschlossen an I2C Bus 0 (Wire0)



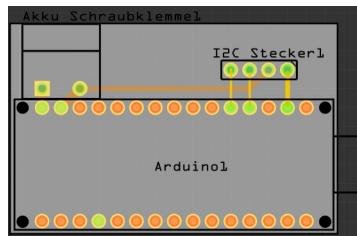
Variante (b): Akku-Anschluss + Arduino + TCS34725 angeschlossen an I2C Bus 1 (Wire1)



So sieht es z.B. auf einem Steckbrett aus:



So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



4.4.3 Spektrometer Farbsensor

Produkt-Link: <https://shop.pimoroni.com/products/as7262-6-channel-spectral-sensor-spectrometer-breakout?variant=13575427260499>

Beispiel-Programm: `Beispiele_06_FarbsensorSpektrometer_06_FarbsensorSpektrometer.ino`

Der Spektrometer-Farbsensor leuchtet die Umgebung mit weißem Licht aus und misst die Intensität des reflektierten Lichts. Licht kann als Welle beschrieben werden. Eine „Farbe“ entspricht der Wellenlänge. Das „Licht-Spektrum“ beschreibt alle möglichen Wellenlängen. Der Sensor misst festgelegte Wellenlängen (450, 500, 550, 570, 600, 650 nm), also bestimmte Farben (violett, blau, grün, gelb, orange, rot).

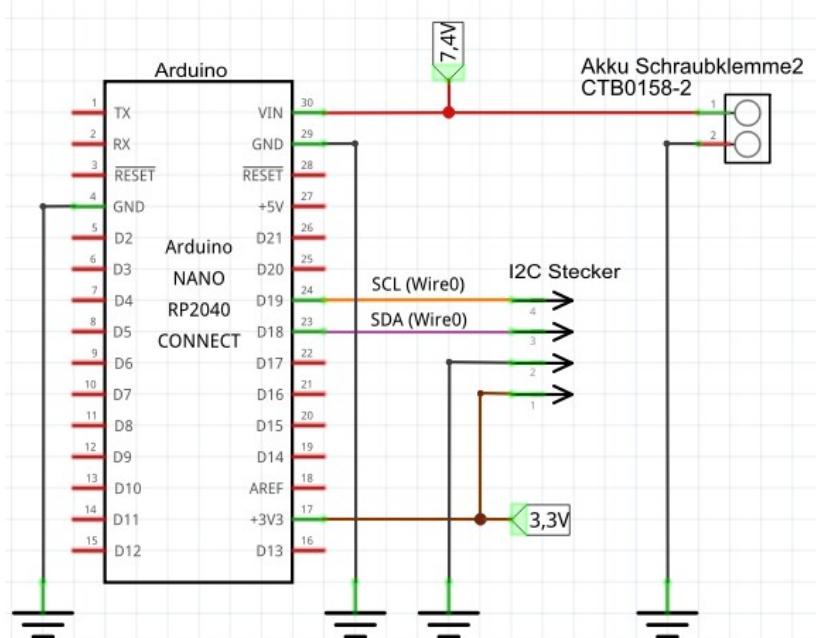
Eigenschaft	Wert	OK?
Logik-Spannung	3.6V	<input checked="" type="checkbox"/> 3,3 V vom Arduino
Stromaufnahme	0,038 A	siehe 1.8 Stromaufnahme
I2C Adresse	0x49	siehe 1.6 I2C Adressen

Pinout:

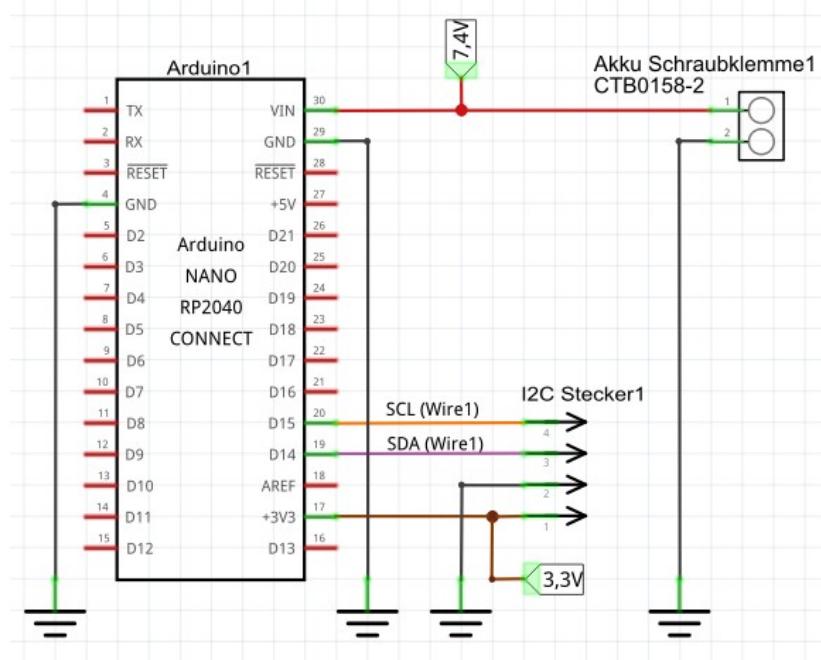


AS7262 Pin	verbinden mit
SDA – I2C Bus Daten Eingang/Ausgang	Arduino SDA
SCL – I2C Bus Takt Eingang	Arduino SCL
3-6V – Logik-Spannung Eingang	Arduino 3,3V
GND – Spannungs-Referenz (Masse)	Akku ⊖ und alle anderen GND
3V3 – 3,3V Ausgang	nichts (n.c.)
INT – kann man optional an irgendeinen Digital-Pin anschließen, wenn man den Sensor effizienter auslesen will	nichts (n.c.)

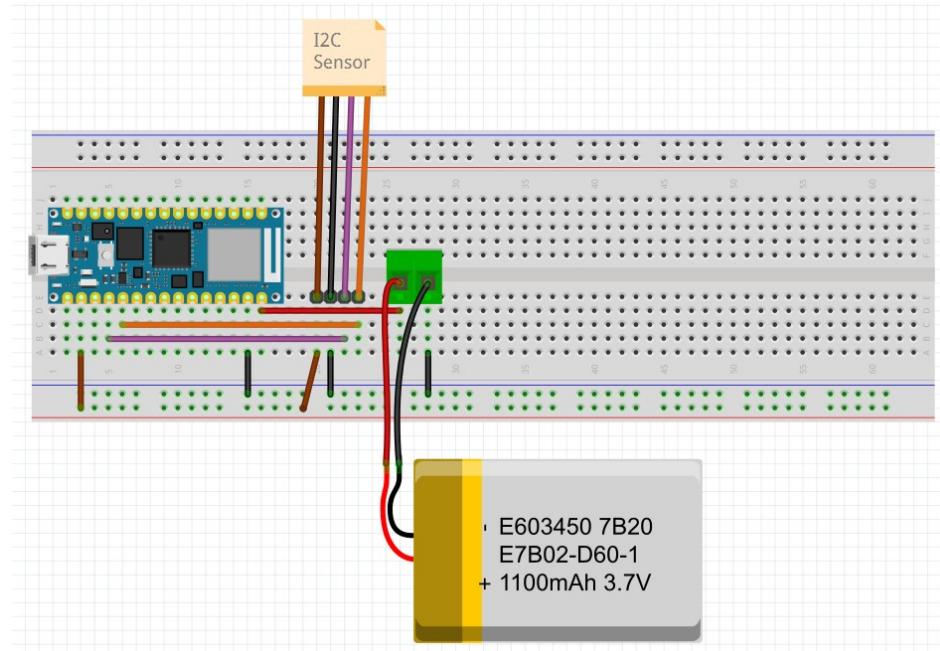
Variante (a): Akku-Anschluss + Arduino + AS7262 angeschlossen an I2C Bus 0 (Wire0)



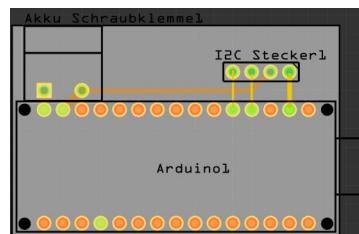
Variante (b): Akku-Anschluss + Arduino + AS7262 angeschlossen an I2C Bus 1 (Wire1)



So sieht es z.B. auf einem Steckbrett aus:



So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



4.4.4 6-Kanal Helligkeitssensor

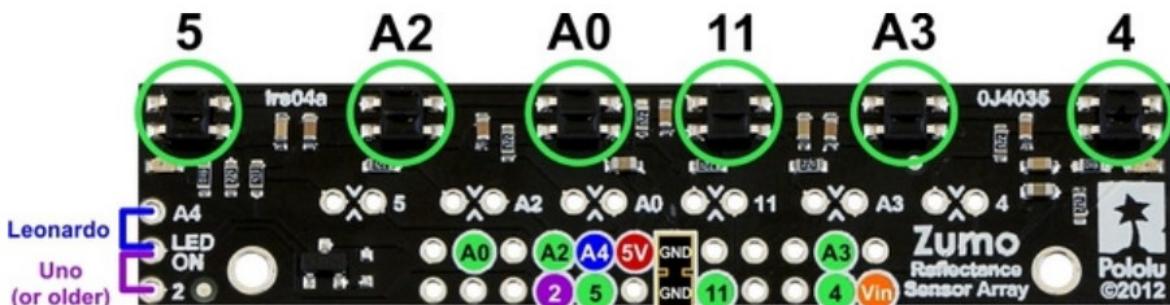
Produkt-Link: <https://www.pololu.com/product/1419>

Beispiel-Programm: Beispiele\10_ReflektionsSensor6Kanal\10_ReflektionsSensor6Kanal.ino

Diese Reflektions-Sensorleiste leuchtet die Umgebung mit Infrarot-Licht aus (für Menschen unsichtbar) und misst die Intensität des reflektierten Lichts. Die Sensorleiste hat 6 Sensoren/Kanäle, aber da jeder Sensor selbst nur 1 Kanal hat, können nur Helligkeiten gemessen werden und keine Farben.

Eigenschaft	Wert	OK?
Logik-Spannung	5V	<input checked="" type="checkbox"/> funktioniert aber auch mit 3,3V vom Arduino
LED-Spannung	7,45V	(a) <input checked="" type="checkbox"/> 7,4 V vom 4.7.1 LiPo Akku (7,4V) : Die LEDs leuchten heller, aber die Helligkeit ändert sich mit dem Ladezustand des Akkus.
		(b) <input checked="" type="checkbox"/> 5 V vom 4.7.5 5V-Spannungsregler (Extern) oder 3.1 Arduino Due : Die LEDs leuchten dunkler, aber die Helligkeit ist immer gleich.
Stromaufnahme	0,24 A	siehe 1.8 Stromaufnahme

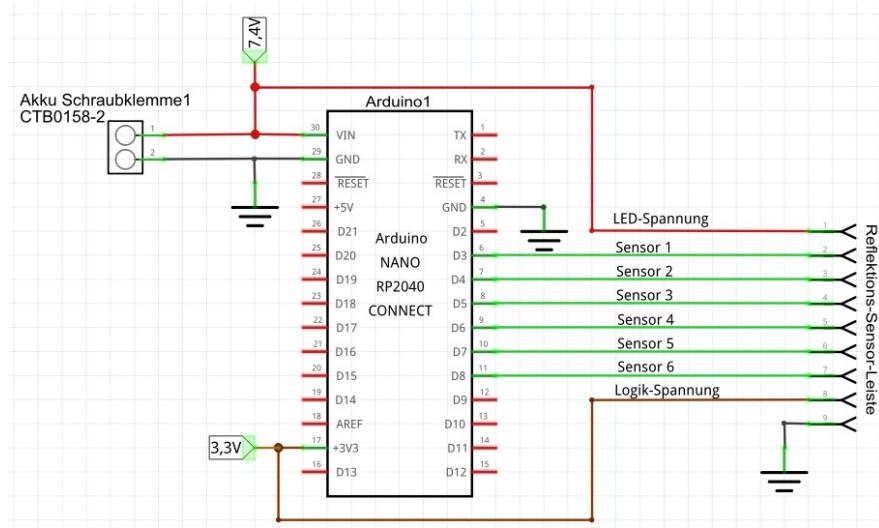
Pinout:



Zumo Reflectance Sensor Array Pin	verbinden mit
5V – Logik-Spannung Eingang	Arduino 3,3V Pin
VIN – LED-Spannung Eingang (Beleuchtung)	Akku \oplus oder 5V Spannungsregler VOUT
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND (1x reicht, der andere GND Pin muss nicht angeschlossen werden)
5 – Sensor ganz rechts (von oben geschaut)	irgendein Arduino Digital In/Out Pin
A2 – Sensor rechts (von oben geschaut)	irgendein Arduino Digital In/Out Pin
A0 – Sensor mittig rechts (von oben geschaut)	irgendein Arduino Digital In/Out Pin
11 – Sensor mittig links (von oben geschaut)	irgendein Arduino Digital In/Out Pin
A3 – Sensor links (von oben geschaut)	irgendein Arduino Digital In/Out Pin
4 – Sensor ganz links (von oben geschaut)	irgendein Arduino Digital In/Out Pin
2 – LEDs abschalten (Variante 1)	nichts (n.c.)
A4 – LEDs abschalten (Variante 1)	nichts (n.c.)

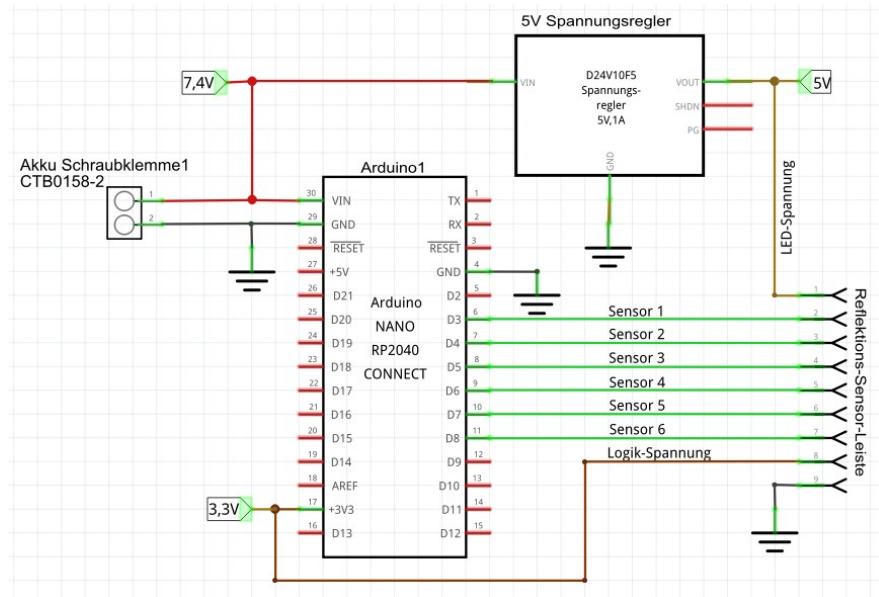
Variante (a): Akku-Anschluss + Arduino + Zumo Reflectance Sensor Array (VIN: 7,4V)

- Da die LEDs viel Strom verbrauchen, müssen sie aus einem 5V-Spannungs-Regler versorgt werden.

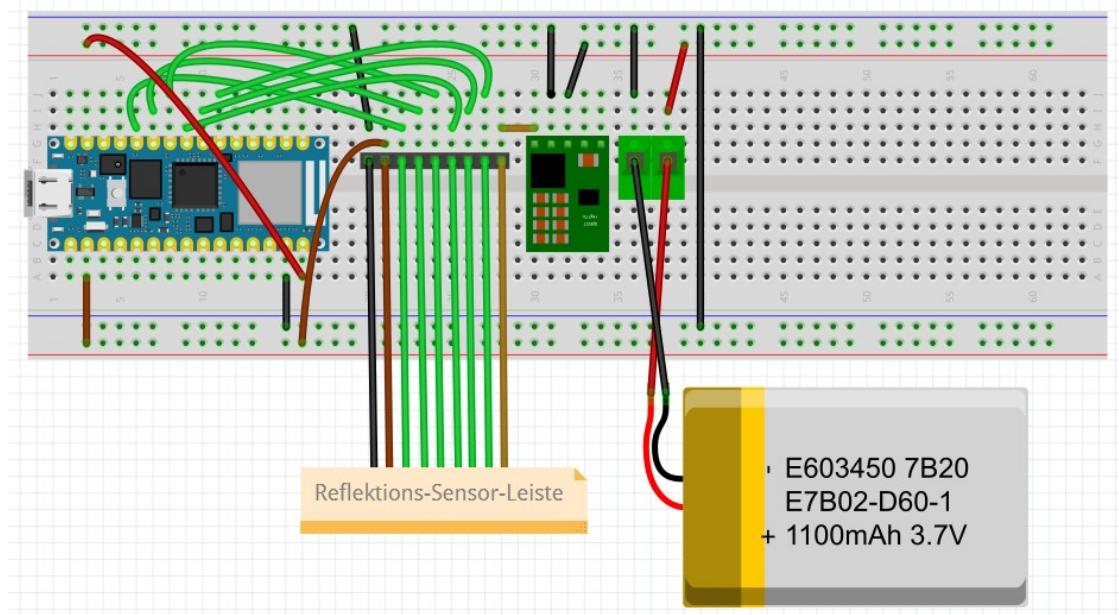


Variante (b): Akku-Anschluss + Arduino + Zumo Reflectance Sensor Array (VIN: 5V)

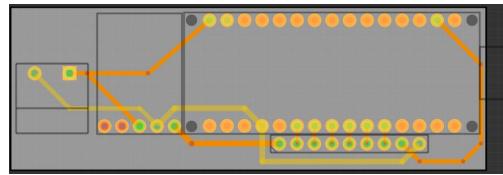
- Da die LEDs viel Strom verbrauchen, müssen sie aus einem 5V-Spannungs-Regler versorgt werden.



So sieht Variante (b) z.B: auf einem Steckbrett aus:



So sieht Variante (b) z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



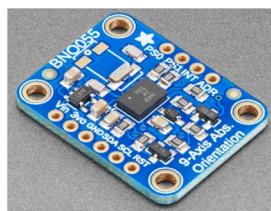
4.5 Bewegungssensoren (IMUs)

4.5.1 9-Achsen IMU (Onboard ZumoShield)

TODO

4.5.2 6-Achsen IMU (Onboard Arduino Nano RP2040 Connect)

4.5.3 9-Achsen IMU mit Sensorfusion



Produkt-Link: <https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview>

Power Pins

- **VIN:** 3.3-5.0V power supply input
- **3VO:** 3.3V output from the on-board linear voltage regulator, you can grab up to about 50mA as necessary
- **GND:** The common/GND pin for power and logic

I2C Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.

Produkt-Link:

Beispiel-Programm:

Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> X 3,3 V vom Arduino

Pinout:

XX Pin	verbinden mit
XX - XX	XX

VCC – Motor-Spannung Eingang	5V Spannungsregler VOUT
GND – Spannungs-Referenz (Masse)	Akku ⊕ und alle anderen GND

Schaltplan: Akku-Anschluss + Arduino + Servo (VCC: 5V)

- Der Servo-Motor beinhaltet bereits die Motorsteuerung, muss aber aus einem 6V-Spannungsregler versorgt werden, da die Akku-Spannung zu hoch ist.

So sieht es z.B. auf einem Steckbrett aus:

So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)

4.6 Abstandssensoren

4.6.1 Mikro-Taster

Produkt-Link:

Beispiel-Programm:

Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> X 3,3 V vom Arduino

Pinout:

XX Pin	verbinden mit
XX – XX	XX
VCC – Motor-Spannung Eingang	5V Spannungsregler VOUT
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND

Schaltplan: Akku-Anschluss + Arduino + Servo (VCC: 5V)

- Der Servo-Motor beinhaltet bereits die Motorsteuerung, muss aber aus einem 6V-Spannungsregler versorgt werden, da die Akku-Spannung zu hoch ist.

So sieht es z.B. auf einem Steckbrett aus:

So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)

4.6.2 1-Kanal Abstandssensor

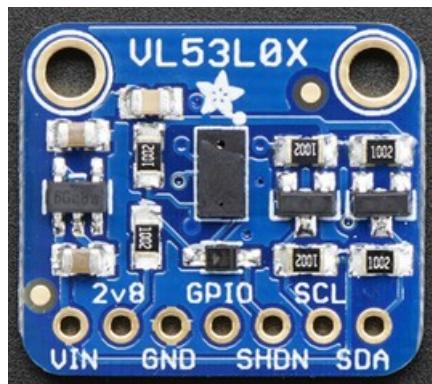
Produkt-Link: <https://www.adafruit.com/product/3317>

Beispiel-Programm: Beispiele\08_AbstandsSensor1Kanal\08_AbstandsSensor1Kanal.ino

Der Time-of-Flight Abstandssensor sendet einen gepulsten Infrarot-Laserstrahl aus (unsichtbar für Menschen) und misst, wie lange es dauert, bis das Signal von einem Objekt zurück reflektiert wird. Das funktioniert mit glänzenden/hellen Objekten besser als mit dunklen/matten. Der Sensor misst also den Abstand mit Licht und hat aufgrund des gebündelten Laser-Strahls eine sehr enge [Keule](#).

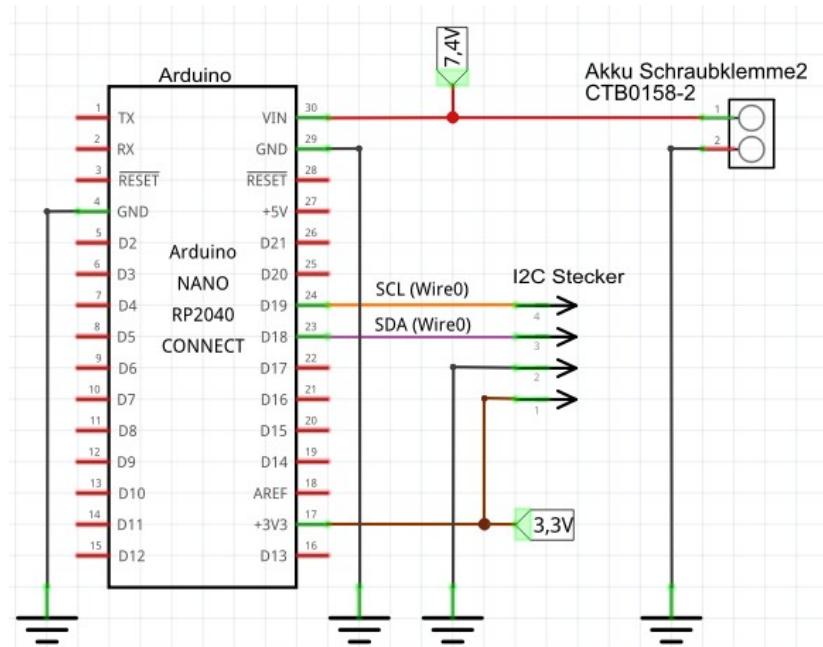
Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> 3,3 V vom Arduino
Stromaufnahme	0,019 A	siehe 1.8 Stromaufnahme
I2C Adresse	0x29	siehe 1.6 I2C Adressen

Pinout:

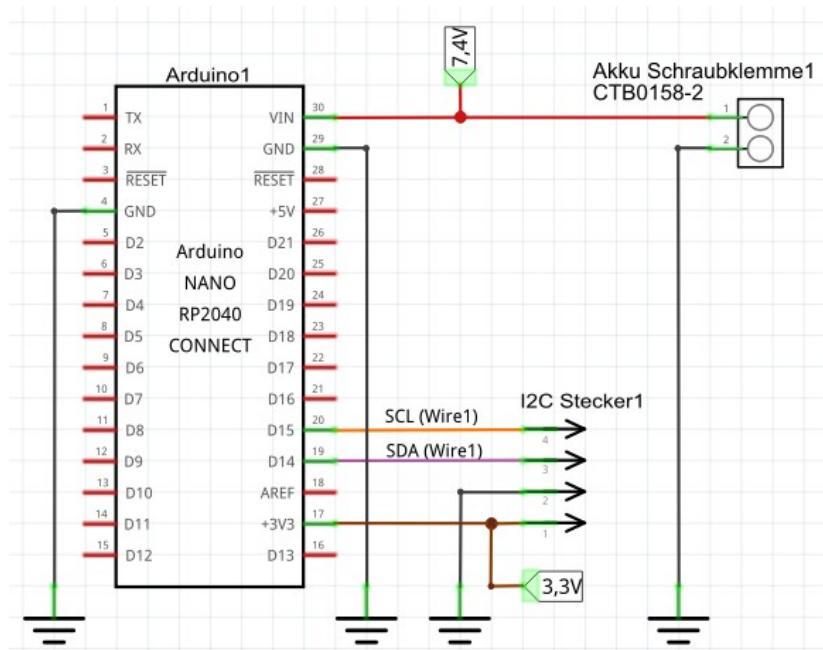


VL53L0X Pin	verbinden mit
SDA – I2C Bus Daten Eingang/Ausgang	Arduino SDA
SCL – I2C Bus Takt Eingang	Arduino SCL
VIN – Logik-Spannung Eingang	Arduino 3,3V
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND
2v8 – 2,8V Ausgang	nichts (n.c.)
SHDN – kann man optional an irgendeinen Digital-Pin anschließen, um den Sensor programmatisch abzuschalten, während man die I2C Adresse eines anderen Sensors auf demselben I2C Bus ändert	nichts (n.c.)
GPIO – kann man optional an irgendeinen Digital-Pin anschließen, wenn man sich vom Sensor benachrichtigen lassen möchten, z.B. wenn der Abstand unter eine bestimmte Schwelle absinkt	nichts (n.c.)

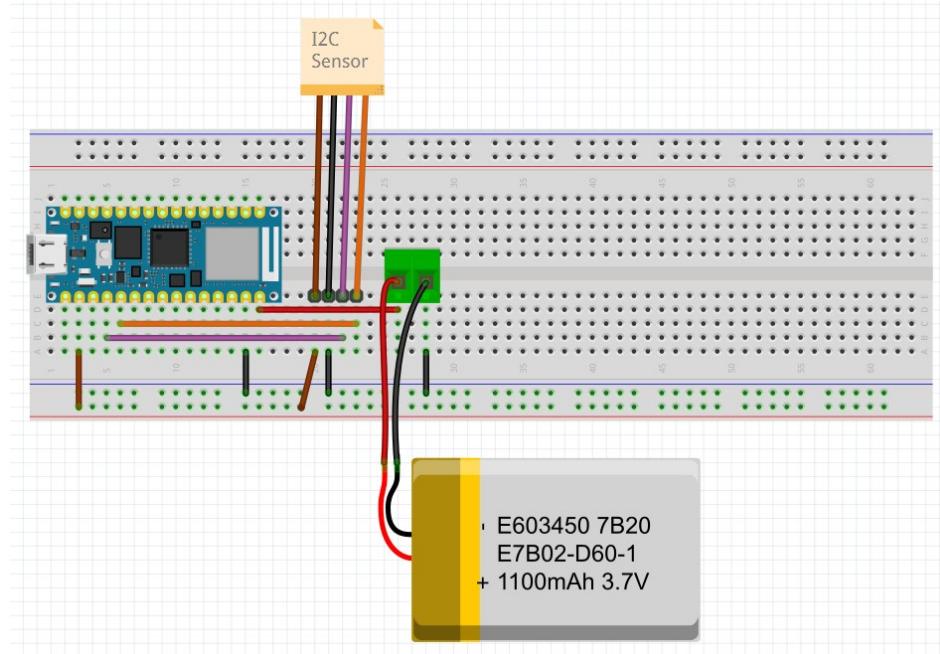
Variante (a): Akku-Anschluss + Arduino + VL53L0X angeschlossen an I2C Bus 0 (Wire0)



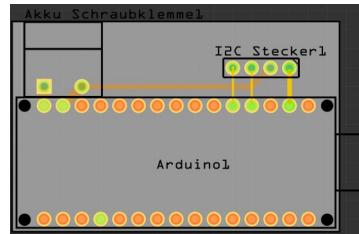
Variante (b): Akku-Anschluss + Arduino + VL53L0X angeschlossen an I2C Bus 1 (Wire1)



So sieht es z.B: auf einem Steckbrett aus:



So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



4.6.3 64-Kanal Abstandssensor

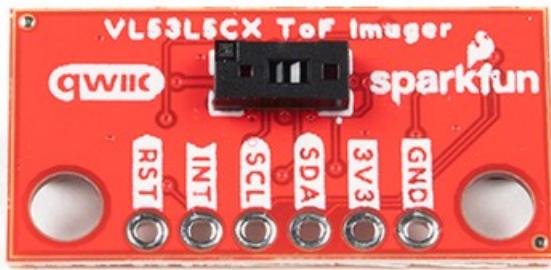
Produkt-Link: <https://www.sparkfun.com/products/19013>

Beispiel-Programm: `Beispiele\09_AbstandsSensor64Kanal_09_AbstandsSensor64Kanal.ino`

Der Time-of-Flight Abstandssensor sendet einen gepulsten Infrarot-Laserstrahl aus (unsichtbar für Menschen) und misst, wie lange es dauert, bis das Signal von einem Objekt zurück reflektiert wird. Das funktioniert mit glänzenden/hellen Objekten besser als mit dunklen/matten. Die Richtung des Laser-Beschusses steuert der Sensor und scannt damit eine Matrix aus 8x8 Punkten im Sichtfeld ab. Daraus ergibt sich eine 3D-Punktwolke aus 64 Abstands-Werten. Es handelt sich also um ein Mini-LiDAR.

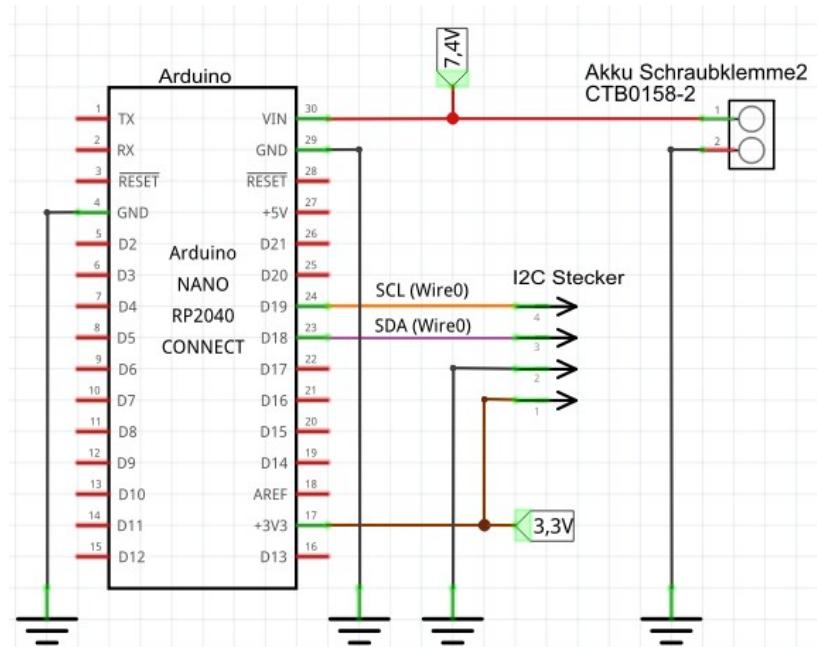
Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> 3,3 V vom Arduino
Stromaufnahme	0,15 A	siehe 1.8 Stromaufnahme
I2C Adresse	0x29	siehe 1.6 I2C Adressen

Pinout:

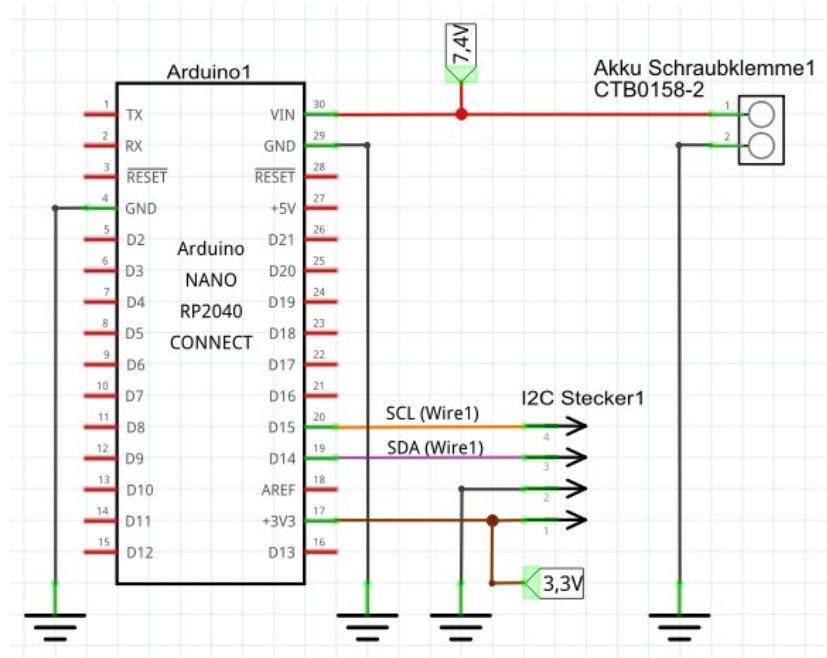


VL53L5CX Pin	verbinden mit
SDA – I2C Bus Daten Eingang/Ausgang	Arduino SDA
SCL – I2C Bus Takt Eingang	Arduino SCL
3V3 – Logik-Spannung Eingang	Arduino 3,3V
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND
2v8 – 2,8V Ausgang	nichts (n.c.)
RST – kann man optional an irgendeinen Digital-Pin anschließen, um den Sensor programmatisch abzuschalten, während man die I2C Adresse eines anderen Sensors auf demselben I2C Bus ändert	nichts (n.c.)
INT – kann man optional an irgendeinen Digital-Pin anschließen, wenn man den Sensor effizienter auslesen will	nichts (n.c.)

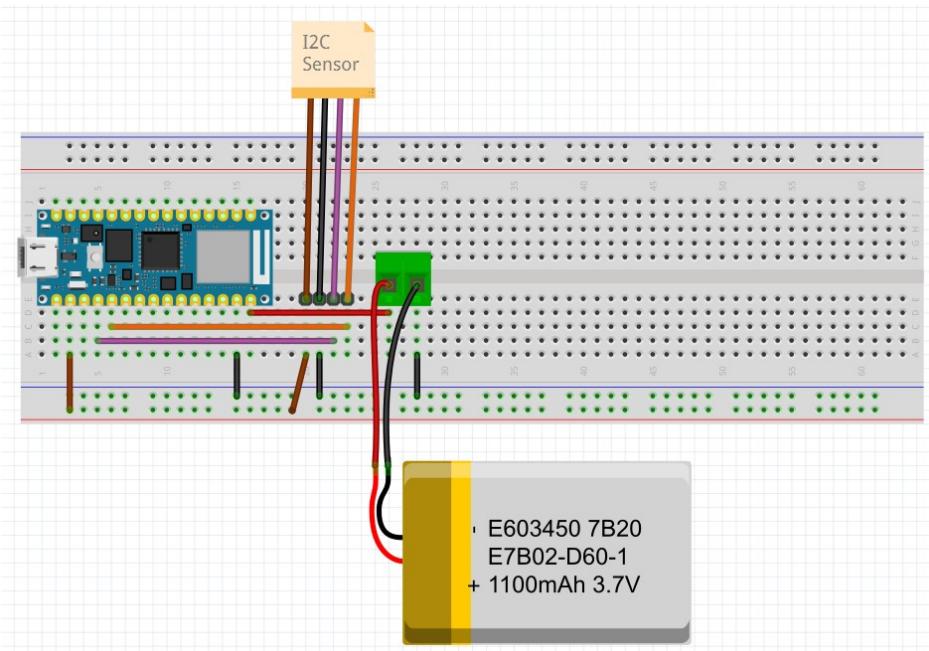
Variante (a): Akku-Anschluss + Arduino + VL53L5CX angeschlossen an I2C Bus 0 (Wire0)



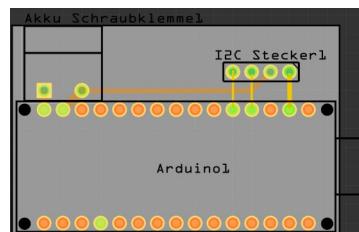
Variante (b): Akku-Anschluss + Arduino + VL53L5CX angeschlossen an I2C Bus 1 (Wire1)



So sieht es z.B. auf einem Steckbrett aus:



So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)



4.7 Spannungsversorgung

4.7.1 LiPo Akku (7,4V)

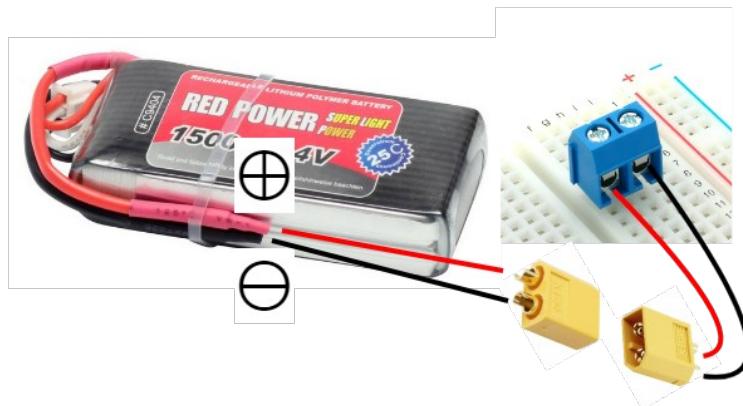
Produkt-Link:

- **LiPo Akku:** <https://www.conrad.de/de/p/red-power-modellbau-akkupack-lipo-7-4-v-1500-mah-zellen-zahl-2-25-c-softcase-offene-kabelenden-1533843.html>
- **Akku-Stecker:** <https://www.conrad.de/de/p/reely-re-6586515-akku-stecker-xt60-3-5-mm-zum-loeten-1-st-2195505.html>
- **Akku Schraubklemme:** z.B. <https://www.berrybase.de/anschlussklemme-schraubbar-900-gewinkelt-1-25mm2-rm-5-08mm-2-polig>

Der Akku ist notwendig, um den Arduino mit Spannung zu versorgen, wenn er nicht via USB an einen Rechner angeschlossen ist (also frei herum fährt). Z.B. DC-Motoren, Servo-Motoren und LEDs verbrauchen viel Strom, die der USB nicht liefern kann. Für Aktoren benötigt man also immer einen Akku.

Eigenschaft	Wert	OK?
Akku-Spannung	4,7V	<input checked="" type="checkbox"/> OK für Arduino, DC-Motoren und VIN Helligkeitssensor <input type="checkbox"/> X nicht OK für Sensoren (3,3V) und Servo-Motoren (5..6V)
max. Strom	37A	<input checked="" type="checkbox"/> so viel werden wie nie brauchen. Überschlagen und stark aufgerundet: (DC-Motoren: 4x1,6A) + (Servos: 4x0,8A) + (2A alles andere) = 11,6A
Stecker		

Pinout:



Akku Pin	verbinden mit
⊕ – Akku 7,4V	via Ein/Aus-Schalter: VIN Spannungsregler , VIN Arduino , VIN Motorsteuerung , VIN Helligkeitssensor
⊖ – Spannungs-Referenz (Masse)	via Ein/Aus-Schalter: alle anderen GND

Schaltplan: Akku-Anschluss + Arduino + Servo (VCC: 5V)

- Der Servo-Motor beinhaltet bereits die Motorsteuerung, muss aber aus einem 6V-Spannungsregler versorgt werden, da die Akku-Spannung zu hoch ist.

So sieht es z.B. auf einem Steckbrett aus:

So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)

4.7.2 3,3V Spannungsregler (Onboard Arduino)

4.7.3 3,3V Spannungsregler (Extern)

Produkt-Link:

Beispiel-Programm:

Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> X 3,3 V vom Arduino

Pinout:

XX Pin	verbinden mit
XX – XX	XX
VCC – Motor-Spannung Eingang	5V Spannungsregler VOUT
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND

Schaltplan: Akku-Anschluss + Arduino + Servo (VCC: 5V)

- Der Servo-Motor beinhaltet bereits die Motorsteuerung, muss aber aus einem 6V-Spannungsregler versorgt werden, da die Akku-Spannung zu hoch ist.

So sieht es z.B. auf einem Steckbrett aus:

So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)

4.7.4 5V Spannungsregler (Onboard Arduino)

4.7.5 5V-Spannungsregler (Extern)

Produkt-Link:

Beispiel-Programm:

Eigenschaft	Wert	OK?
Logik-Spannung	3,3..5V	<input checked="" type="checkbox"/> X 3,3 V vom Arduino

Pinout:

XX Pin	verbinden mit
XX – XX	XX
VCC – Motor-Spannung Eingang	5V Spannungsregler VOUT
GND – Spannungs-Referenz (Masse)	Akku \ominus und alle anderen GND

Schaltplan: Akku-Anschluss + Arduino + Servo (VCC: 5V)

- Der Servo-Motor beinhaltet bereits die Motorsteuerung, muss aber aus einem 6V-Spannungsregler versorgt werden, da die Akku-Spannung zu hoch ist.

So sieht es z.B. auf einem Steckbrett aus:

So sieht es z.B. auf einer Platine aus: (Original-Größe, also 1:1, rein zoomen für Detail)

4.8 Sonstiges

4.8.1 I2C Multiplexer

Produkt-Link: <https://www.berrybase.de/tca9548a-i2c-multiplexer-breakout-board>

TODO

4.8.2 Jumper-Kabel

4.8.3 Schalter

4.8.4 Platinen-Abstandshalter



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20